RNA-seq workflow - gene-level exploratory analysis and differential expression

CSAMA2016 version

July 10, 2016

Abstract

Here we walk through an end-to-end gene-level RNA-seq differential expression workflow using Bioconductor packages. We will start from the FASTQ files, show how these were aligned to the reference genome, and prepare a count matrix which tallies the number of RNA-seq reads/fragments within each gene for each sample. We will perform exploratory data analysis (EDA) for quality assessment and to explore the relationship between samples, perform differential gene expression analysis, and visually explore the results.

Citing scientific research software

If you use the results from an R analysis package in published research, you can find the proper citation for the software by typing `citation("pkgName")` , where you would substitute the name of the package for `pkgName` . Citing methods papers helps to support and reward the individuals who put time into open source software for genomic data analysis.

Introduction

Bioconductor has many packages which support analysis of high-throughput sequence data, including RNA sequencing (RNA-seq). The packages which we will use in this workflow include core packages maintained by the Bioconductor core team for importing and processing raw sequencing data and loading gene annotations. We will also use contributed packages for statistical analysis and visualization of sequencing data. Through scheduled releases every 6 months, the Bioconductor project ensures that all the packages within a release will work together in harmony (hence the "conductor" metaphor). The packages used in this workflow are loaded with the *library* function and can be installed by following the Bioconductor package installation instructions (http://bioconductor.org/install/#install-bioconductor-packages).

A published version of this workflow, including reviewer reports and comments is available at F1000Research (http://f1000research.com/articles/4-1070) (Love et al. 2015),.

If you have questions about this workflow or any Bioconductor software, please post these to the Bioconductor support site (https://support.bioconductor.org/). If the questions concern a specific package, you can tag the post with the name of the package, or for general questions about the workflow, tag the post with `rnaseqgene` . Note the posting guide (http://www.bioconductor.org/help/support/posting-guide/) for crafting an optimal question for the support site.

Experimental data

The data used in this workflow is stored in the *airway (http://bioconductor.org/packages/airway)* package that summarizes an RNA-seq experiment wherein airway smooth muscle cells were treated with dexamethasone, a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014). Glucocorticoids are used, for example, by people with asthma to reduce inflammation of the airways. In the experiment, four human airway smooth muscle cell lines were treated with 1 micromolar dexamethasone for 18 hours. For each of the four cell lines, we have a treated and an untreated sample. For more description of the experiment see the article, PubMed entry 24926665 (http://www.ncbi.nlm.nih.gov/pubmed/24926665), and for raw data see the GEO entry GSE52778 (http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE52778).

File failed to load: /extensions/MathZoom.js

Goal of this workflow

Our goal in this workflow is to bring a summary of the RNA-seq experiment into R/Bioconductor for visualization and statistical testing. We want to visualize the relationships between the samples (within and across the treatment), and then we want to perform statistical tests to find which genes are changing their expression due to treatment.

An overview of the steps we will take (and alternatives) is:

1. Preprocess FASTQ files
   - Align to the genome with STAR or other alignment tools.
   - *or:* Quantify at transcript level using Sailfish, Salmon or kallisto (not covered in this workflow).
2. Summarize into a gene-level count matrix
   - Count number of aligned fragments that can be unambiguously assigned to genes.
   - *or:* Use the *tximport (http://bioconductor.org/packages/tximport)* package to import transcript quantifications and summarize to the gene level (not covered in this workflow).
3. Convert the count matrix into a package-specific object, e.g. a *DESeqDataSet* for DESeq2 or a *DGEList* for edgeR.
4. Make exploratory plots, such as PCA plots and sample-sample distance plots.
5. Perform differential expression testing for all genes.
6. Make summary plots of the differential expression results.

Summarizing an RNA-seq experiment as a count matrix

The count-based statistical methods, such as *DESeq2 (http://bioconductor.org/packages/DESeq2)* (Love, Huber, and Anders 2014), *edgeR (http://bioconductor.org/packages/edgeR)* (M. D. Robinson, McCarthy, and Smyth 2009), *limma (http://bioconductor.org/packages/limma)* with the voom method (Law et al. 2014), *DSS (http://bioconductor.org/packages/DSS)* (H. Wu, Wang, and Wu 2013), *EBSeq (http://bioconductor.org/packages/EBSeq)* (Leng et al. 2013) and *BaySeq (http://bioconductor.org/packages/BaySeq)* (Hardcastle and Kelly 2010), expect input data as obtained, e.g., from RNA-seq or another high-throughput sequencing experiment, in the form of a matrix of integer values, or "counts". The value in the *i*-th row and the *j*-th column of the matrix tells how many reads (or fragments, for paired-end RNA-seq) have been unambiguously assigned to gene *i* in sample *j*. Analogously, for other types of assays, the rows of the matrix might correspond e.g., to binding regions (with ChIP-Seq), species of bacteria (with metagenomic datasets), or peptide sequences (with quantitative mass spectrometry).

The values in the matrix are counts of sequencing reads (in the case of single-end sequencing) or fragments (for paired-end sequencing). This is important for the count-based statistical models, e.g. *DESeq2* or *edgeR*, as only the counts allow assessing the measurement precision correctly. It is important to *never* provide counts that were normalized for sequencing depth/library size, as the statistical model is most powerful when applied to counts, and is designed to account for library size differences internally.

As we will discuss later, an alternative to using raw counts of reads or fragments aligned to the genome is to use *estimated* counts from software that use *pseudo-alignment* to the *transcriptome* (Soneson, Love, and Robinson 2015).

Aligning reads to a reference genome

File failed to load: /extensions/MathZoom.js

The computational analysis of an RNA-seq experiment often begins earlier: we first obtain a set of FASTQ files that contain the nucleotide sequence of each read and a quality score at each position. These reads must first be aligned to a reference genome or transcriptome. It is important to know if the sequencing experiment was single-end or paired-end, as the alignment software will require the user to specify both FASTQ files for a paired-end experiment. The output of this alignment step is commonly stored in a file format called SAM/BAM (http://samtools.github.io/hts-specs).

A number of software programs exist to align reads to a reference genome, and the development is too rapid for this document to provide an up-to-date list. We recommend consulting benchmarking papers that discuss the advantages and disadvantages of each software, which include accuracy, sensitivity in aligning reads over splice junctions, speed, memory required, usability, and many other features.

The reads for this experiment were aligned to the Ensembl release 75 (Flicek et al. 2014) human reference genome using the STAR spliced read aligner (https://code.google.com/p/rna-star/) (Dobin et al. 2013). In this example, we have a file in the current directory called `files` with each line containing an identifier for each experiment, and we have all the FASTQ files in a subdirectory `fastq`. If you have downloaded the FASTQ files from the Sequence Read Archive, the identifiers would be SRA run IDs, e.g. `SRR1039520`. You should have two files for a paired-end experiment for each ID, `fastq/SRR1039520_1.fastq1` and `fastq/SRR1039520_2.fastq`, which give the first and second read for the paired-end fragments. If you have performed a single-end experiment, you would only have one file per ID. We have also created a subdirectory, `aligned`, where STAR will output its alignment files.

The following chunk of code was run on the command line (outside of R) to align the paired-end reads to the genome:

```
for f in `cat files`; do STAR --genomeDir ../STAR/ENSEMBL.homo_sapiens.release-75 \
--readFilesIn fastq/$f\_1.fastq fastq/$f\_2.fastq \
--runThreadN 12 --outFileNamePrefix aligned/$f.; done
```

For the latest versions of STAR, the flag `--outSAMtype BAM SortedByCoordinate` can be added to automatically sort the aligned reads and turn them into compressed BAM files. The BAM files for a number of sequencing runs can then be used to generate count matrices, as described in the following section.

Locating BAM files and the sample table

Besides the count matrix that we will use later, the *airway (http://bioconductor.org/packages/airway)* package also contains eight BAM files with a small subset of reads from the experiment – enough for us to try out counting reads for a small set of genes.

The reads were selected which aligned to a small region of chromosome 1. We chose a subset of reads because the full alignment files are large (a few gigabytes each), and because it takes 10-30 minutes to count the full set of fragments for each sample. We will use these files to demonstrate how a count matrix can be constructed from BAM files. Afterwards, we will load the full count matrix corresponding to all samples and all data, which is already provided in the same package, and will continue the analysis with that full matrix.

We load the data package with the example data:

```
library("airway")
```

The R function *system.file* can be used to find out where on your computer the files from a package have been installed. Here we ask for the full path to the `extdata` directory, where the external data that is part of the *airway (http://bioconductor.org/packages/airway)* package has been stored.

Note that the use of *system.file* is particular to this workflow, because we have the data stored in an R package. You would not typically use this function for your own pipeline with data stored in directories on your local machine or cluster.

```
dir <- system.file("extdata", package="airway", mustWork=TRUE)
```

In this directory, we find the eight BAM files (and some other files):

```
list.files(dir)
```

```
##  [1] "GSE52778_series_matrix.txt"        "Homo_sapiens.GRCh37.75_subset.gtf"
##  [3] "sample_table.csv"                  "SraRunInfo_SRP033351.csv"
##  [5] "SRR1039508_subset.bam"             "SRR1039509_subset.bam"
##  [7] "SRR1039512_subset.bam"             "SRR1039513_subset.bam"
##  [9] "SRR1039516_subset.bam"             "SRR1039517_subset.bam"
## [11] "SRR1039520_subset.bam"             "SRR1039521_subset.bam"
```

Typically, we have a table with detailed information for each of our samples that links samples to the associated FASTQ and BAM files. For your own project, you might create such a comma-separated value (CSV) file using a text editor or spreadsheet software such as Excel.

We load such a CSV file with *read.csv*:

```
csvfile <- file.path(dir,"sample_table.csv")
sampleTable <- read.csv(csvfile,row.names=1)
sampleTable
```

```
##            SampleName    cell   dex albut        Run avgLength Experiment    Sample
  BioSample
## SRR1039508 GSM1275862  N61311 untrt untrt SRR1039508       126  SRX384345 SRS508568 S
AMN02422669
## SRR1039509 GSM1275863  N61311   trt untrt SRR1039509       126  SRX384346 SRS508567 S
AMN02422675
## SRR1039512 GSM1275866 N052611 untrt untrt SRR1039512       126  SRX384349 SRS508571 S
AMN02422678
## SRR1039513 GSM1275867 N052611   trt untrt SRR1039513        87  SRX384350 SRS508572 S
AMN02422670
## SRR1039516 GSM1275870 N080611 untrt untrt SRR1039516       120  SRX384353 SRS508575 S
AMN02422682
## SRR1039517 GSM1275871 N080611   trt untrt SRR1039517       126  SRX384354 SRS508576 S
AMN02422673
## SRR1039520 GSM1275874 N061011 untrt untrt SRR1039520       101  SRX384357 SRS508579 S
AMN02422683
## SRR1039521 GSM1275875 N061011   trt untrt SRR1039521        98  SRX384358 SRS508580 S
AMN02422677
```

File failed to load: /extensions/MathZoom.js

Once the reads have been aligned, there are a number of tools that can be used to count the number of reads/fragments that can be assigned to genomic features for each sample. These often take as input SAM/BAM alignment files and a file specifying the genomic features, e.g. a GFF3 or GTF file specifying the gene models.

tximport: transcript abundance summarized to gene-level

An alternative to the alignment-counting workflow is the *tximport* workflow, which leverages transcript quantification methods such as *Sailfish* (Patro, Mount, and Kingsford 2014), *Salmon* (Patro, Duggal, and Kingsford 2015), *kallisto* (Bray et al. 2015), and *RSEM* (Li and Dewey 2011), to estimate abundances without aligning reads (so skipping the generation of BAM files). Advantages of using *tximport* to produce gene-level count matrices and normalizing offsets, are:

1. This approach corrects for potential changes in gene length across samples (e.g. from differential isoform usage) (Trapnell et al. 2013).
2. Some of these methods are substantially faster and require less memory and disk usage compared to alignment-based methods
3. It is possible to avoid discarding those fragments that can align to multiple genes with homologous sequence (Robert and Watson 2015). Assigning these genes probabilistically and reading in *estimated* counts may increase sensitivity.

For more details and example code, see the manuscript describing this approach (Soneson, Love, and Robinson 2015) and the *tximport (http://bioconductor.org/packages/tximport)* package vignette.

Preparing count matrices from BAM files

The following tools can be used to generate count matrices from reads aligned to the genome:

- *summarizeOverlaps* from *GenomicAlignments (http://bioconductor.org/packages/GenomicAlignments)* (Lawrence et al. 2013)
- *featureCounts* from *Rsubread (http://bioconductor.org/packages/Rsubread)* (Liao, Smyth, and Shi 2014)
- *htseq-count* from HTSeq (http://www-huber.embl.de/users/anders/HTSeq) (Anders, Pyl, and Huber 2015)

Each have slightly different output, which can be gathered into a count matrix. *summarizeOverlaps* produces a *SummarizedExperiment* object, which will be discussed below. *featureCounts* produces a count matrix, and *htseq-count* produces a file for each sample which contains the counts per gene.

We will first demonstrate using the *summarizeOverlaps* method of counting. Using the `Run` column in the sample table, we construct the full paths to the files we want to perform the counting operation on:

```
filenames <- file.path(dir, paste0(sampleTable$Run, "_subset.bam"))
file.exists(filenames)
```

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

We indicate in Bioconductor that these files are BAM files using the *BamFileList* function from the *Rsamtools (http://bioconductor.org/packages/Rsamtools)* package that provides an R interface to BAM files. Here we also specify details about how the BAM files should be treated, e.g., only process 2 million reads at a time. See `?BamFileList` for more information.

File failed to load: /extensions/MathZoom.js

```
library("Rsamtools")
bamfiles <- BamFileList(filenames, yieldSize=2000000)
```

Note: make sure that the chromosome names of the genomic features in the annotation you use are consistent with the chromosome names of the reference used for read alignment. Otherwise, the scripts might fail to count any reads to features due to the mismatching names. For example, a common mistake is when the alignment files contain chromosome names in the style of “ `1` ” and the gene annotation in the style of “ `chr1` ”, or the other way around. See the *seqlevelsStyle* function in the *GenomeInfoDb (http://bioconductor.org/packages/GenomeInfoDb)* package for solutions. We can check the chromosome names (here called “seqnames”) in the alignment files like so:

```
seqinfo(bamfiles[1])
```

```
## Seqinfo object with 84 sequences from an unspecified genome:
##   seqnames   seqlengths isCircular genome
##   1           249250621       <NA>   <NA>
##   10          135534747       <NA>   <NA>
##   11          135006516       <NA>   <NA>
##   12          133851895       <NA>   <NA>
##   13          115169878       <NA>   <NA>
##   ...               ...        ...    ...
##   GL000210.1      27682       <NA>   <NA>
##   GL000231.1      27386       <NA>   <NA>
##   GL000229.1      19913       <NA>   <NA>
##   GL000226.1      15008       <NA>   <NA>
##   GL000207.1       4262       <NA>   <NA>
```

Defining gene models

Next, we need to read in the gene model that will be used for counting reads/fragments. We will read the gene model from an Ensembl GTF file (http://www.ensembl.org/info/website/upload/gff.html) (Flicek et al. 2014). GTF files can be downloaded from Ensembl’s FTP site (http://www.ensembl.org/info/data/ftp/) or other gene model repositories.

featureCounts and *htseq-count* will simply need to know the location of the GTF file, but for *summarizeOverlaps* we first need to create an R object that records the location of the exons for each gene. We first therefore create a *TxDb* (short for “transcript database”), using *makeTxDbFromGFF* from the *GenomicFeatures (http://bioconductor.org/packages/GenomicFeatures)* package. A *TxDb* object is a database that can be used to generate a variety of range-based objects, such as exons, transcripts, and genes. We want to make a list of exons grouped by gene for counting reads or fragments.

There are other options for constructing a *TxDb*. For the *known genes* track from the UCSC Genome Browser (Kent et al. 2002), one can use the pre-built Transcript DataBase: *TxDb.Hsapiens.UCSC.hg19.knownGene (http://bioconductor.org/packages/TxDb.Hsapiens.UCSC.hg19.knownGene)*. If the annotation file is accessible from *AnnotationHub (http://bioconductor.org/packages/AnnotationHub)* (as is the case for the Ensembl genes), a pre-scanned GTF file can be imported using *makeTxDbFromGRanges*. Finally, the *makeTxDbFromBiomart* function can be used to automatically pull a gene model from Biomart using *biomaRt (http://bioconductor.org/packages/biomaRt)* (Durinck et al. 2009).

File failed to load: /extensions/MathZoom.js

Here we will demonstrate loading from a GTF file:

```
library("GenomicFeatures")
gtffile <- file.path(dir,"Homo_sapiens.GRCh37.75_subset.gtf")
txdb <- makeTxDbFromGFF(gtffile, format="gtf")
```

The following line produces a *GRangesList* of all the exons grouped by gene (Lawrence et al. 2013). Each element of the list is a *GRanges* object of the exons for a gene.

```
ebg <- exonsBy(txdb, by="gene")
ebg
```

```
## GRangesList object of length 20:
## $ENSG00000009724
## GRanges object with 18 ranges and 2 metadata columns:
##        seqnames               ranges strand |   exon_id       exon_name
##           <Rle>            <IRanges>  <Rle> | <integer>     <character>
##    [1]        1 [11086580, 11087705]      - |        98 ENSE00000818830
##    [2]        1 [11090233, 11090307]      - |        99 ENSE00000472123
##    [3]        1 [11090805, 11090939]      - |       100 ENSE00000743084
##    [4]        1 [11094885, 11094963]      - |       101 ENSE00000743085
##    [5]        1 [11097750, 11097868]      - |       102 ENSE00003482788
##    ...      ...                  ...    ... .       ...             ...
##   [14]        1 [11106948, 11107176]      - |       111 ENSE00003467404
##   [15]        1 [11106948, 11107176]      - |       112 ENSE00003489217
##   [16]        1 [11107260, 11107280]      - |       113 ENSE00001833377
##   [17]        1 [11107260, 11107284]      - |       114 ENSE00001472289
##   [18]        1 [11107260, 11107290]      - |       115 ENSE00001881401
##
## ...
## <19 more elements>
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Note that the output here just shows the exons of the first gene. The `ebg` object contains 19 more genes, ie., for all the 20 genes descibed in our (very short) example GTF file.

Counting with summarizeOverlaps

After these preparations, the actual counting is easy. The function *summarizeOverlaps* from the *GenomicAlignments (http://bioconductor.org/packages/GenomicAlignments)* package will do this. This produces a *SummarizedExperiment* object that contains a variety of information about the experiment, and will be described in more detail below.

Note: If it is desired to perform counting using multiple cores, one can use the *register* and *MulticoreParam* or *SnowParam* functions from the *BiocParallel (http://bioconductor.org/packages/BiocParallel)* package before the counting call below. Expect that the `summarizeOverlaps` call will take at least 30 minutes per file for a human RNA-seq file with 30 million aligned reads. By sending the files to separate cores, one can speed up the entire counting process.

```
library("GenomicAlignments")
library("BiocParallel")
```

Most modern laptops have CPUs have with two or four cores, hence we gain a bit of speed by asking to use two cores. We could have also skipped this line and the counting step would run in serial.

```
register(MulticoreParam(2))
```

The following call creates the *SummarizedExperiment* object with counts:

```
se <- summarizeOverlaps(features=ebg,
                        reads=bamfiles,
                        mode="Union",
                        singleEnd=FALSE,
                        ignore.strand=TRUE,
                        fragments=TRUE )
```

And let's quickly see what we get, before we explain all the arguments:

```
se
```

```
## class: RangedSummarizedExperiment
## dim: 20 8
## metadata(0):
## assays(1): counts
## rownames(20): ENSG00000009724 ENSG00000116649 ... ENSG00000271794 ENSG00000271895
## rowData names(0):
## colnames(8): SRR1039508_subset.bam SRR1039509_subset.bam ... SRR1039520_subset.bam
##   SRR1039521_subset.bam
## colData names(0):
```

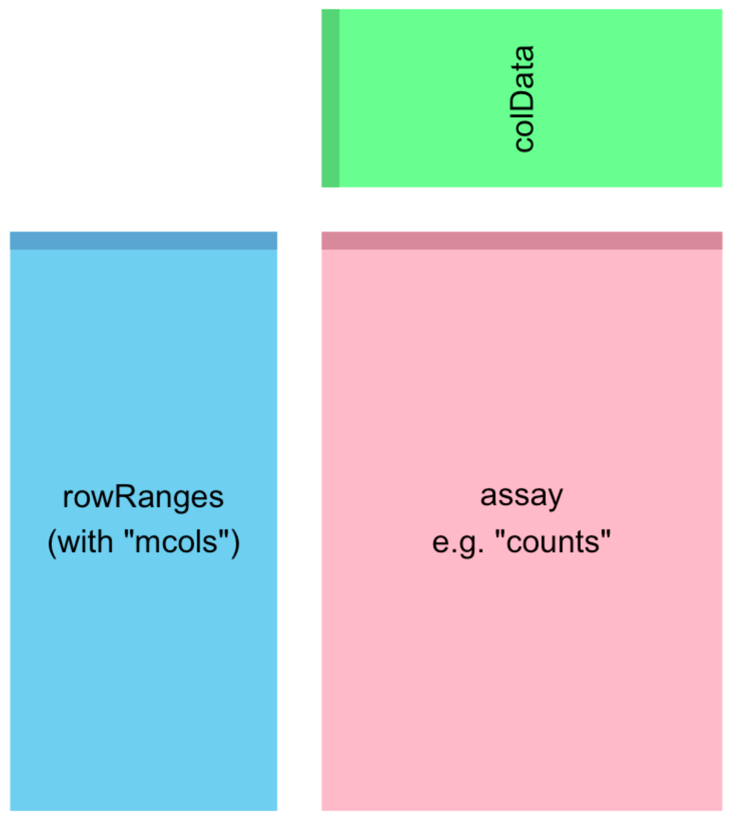
We specify a number of arguments besides the `features` and the `reads`. The `mode` argument describes what kind of read overlaps will be counted. These modes are shown in Figure 1 of the *Counting reads with summarizeOverlaps* vignette for the *GenomicAlignments (http://bioconductor.org/packages/GenomicAlignments)* package. Note that fragments will be counted only once to each gene, even if they overlap multiple exons of a gene which may themselves be overlapping. Setting `singleEnd` to `FALSE` indicates that the experiment produced paired-end reads, and we want to count a pair of reads (a fragment) only once toward the count for a gene. The `fragments` argument can be used when `singleEnd=FALSE` to specify if unpaired reads should be counted (yes if `fragments=TRUE`).

In order to produce correct counts, it is important to know if the RNA-seq experiment was strand-specific or not. This experiment was not strand-specific so we set `ignore.strand` to `TRUE`. However, certain strand-specific protocols could have the reads align only to the opposite strand of the genes. The user must check if the experiment was strand-specific and if so, whether the reads should align to the forward or reverse strand of the genes. For various counting/quantifying tools, one specifies counting on the forward or reverse strand in different ways, although this task is currently easiest with *htseq-count*, *featureCounts*, or the transcript abundance quantifiers mentioned previously. It is always a good idea to check the column sums of the count matrix (see below) to make sure these totals match the expected of the number of reads or fragments aligning to genes. Additionally, one can visually check the read alignments using a genome visualization tool.

SummarizedExperiment

File failed to load: /extensions/MathZoom.js

We will now explain all the parts of the object we obtained from *summarizeOverlaps*. The following figure illustrates the structure of a *SummarizedExperiment* object.



The `assay` (pink block) contains the matrix of counts, the `rowRanges` (blue block) contains information about the genomic ranges, and the `colData` (green block) contains information about the samples. The highlighted line in each block represents the first row (note that the first row of `colData` lines up with the first column of the `assay`).

The *SummarizedExperiment* container is diagrammed in the Figure above and discussed in the latest Bioconductor paper (Huber et al. 2015). In our case we have created a single matrix named "counts" that contains the fragment counts for each gene and sample. The component parts of the *SummarizedExperiment* are accessed with an R function of the same name: `assay` (or `assays`), `rowRanges` and `colData`.

The counts are accessed using `assay`:

```
head( assay(se) )
```

File failed to load: /extensions/MathZoom.js

```
##                 SRR1039508_subset.bam SRR1039509_subset.bam SRR1039512_subset.bam
## ENSG00000009724                    38                    28                    66
## ENSG00000116649                  1004                  1255                  1122
## ENSG00000120942                   218                   256                   233
## ENSG00000120948                  2751                  2080                  3353
## ENSG00000171819                     4                    50                    19
## ENSG00000171824                   869                  1075                  1115
##                 SRR1039513_subset.bam SRR1039516_subset.bam SRR1039517_subset.bam
## ENSG00000009724                    24                    42                    41
## ENSG00000116649                  1313                  1100                  1879
## ENSG00000120942                   252                   269                   465
## ENSG00000120948                  1614                  3519                  3716
## ENSG00000171819                   543                     1                    10
## ENSG00000171824                  1051                   944                  1405
##                 SRR1039520_subset.bam SRR1039521_subset.bam
## ENSG00000009724                    47                    36
## ENSG00000116649                   745                  1536
## ENSG00000120942                   207                   400
## ENSG00000120948                  2220                  1990
## ENSG00000171819                    14                  1067
## ENSG00000171824                   748                  1590
```

We can ask the dimension of the *SummarizedExperiment* (the dimension of the assay matrix), simply with `dim`:

```
dim(se)
```

```
## [1] 20  8
```

```
nrow(se)
```

```
## [1] 20
```

```
ncol(se)
```

```
## [1] 8
```

As we see, in this experiment there are 20 genes and 8 samples. It is also possible to store multiple matrices in a *SummarizedExperiment* object, and to access them with `assays`.

The `rowRanges` for our object is the *GRangesList* we used for counting (one *GRanges* of exons for each row of the count matrix).

```
rowRanges(se)
```

File failed to load: /extensions/MathZoom.js

```
## GRangesList object of length 20:
## $ENSG00000009724
## GRanges object with 18 ranges and 2 metadata columns:
##        seqnames               ranges strand |   exon_id       exon_name
##           <Rle>            <IRanges>  <Rle> | <integer>     <character>
##    [1]        1 [11086580, 11087705]      - |        98 ENSE00000818830
##    [2]        1 [11090233, 11090307]      - |        99 ENSE00000472123
##    [3]        1 [11090805, 11090939]      - |       100 ENSE00000743084
##    [4]        1 [11094885, 11094963]      - |       101 ENSE00000743085
##    [5]        1 [11097750, 11097868]      - |       102 ENSE00003482788
##    ...      ...                  ...    ... .       ...             ...
##   [14]        1 [11106948, 11107176]      - |       111 ENSE00003467404
##   [15]        1 [11106948, 11107176]      - |       112 ENSE00003489217
##   [16]        1 [11107260, 11107280]      - |       113 ENSE00001833377
##   [17]        1 [11107260, 11107284]      - |       114 ENSE00001472289
##   [18]        1 [11107260, 11107290]      - |       115 ENSE00001881401
##
## ...
## <19 more elements>
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
length(rowRanges(se))
```

```
## [1] 20
```

```
rowRanges(se)[[1]]
```

```
## GRanges object with 18 ranges and 2 metadata columns:
##        seqnames               ranges strand |   exon_id       exon_name
##           <Rle>            <IRanges>  <Rle> | <integer>     <character>
##    [1]        1 [11086580, 11087705]      - |        98 ENSE00000818830
##    [2]        1 [11090233, 11090307]      - |        99 ENSE00000472123
##    [3]        1 [11090805, 11090939]      - |       100 ENSE00000743084
##    [4]        1 [11094885, 11094963]      - |       101 ENSE00000743085
##    [5]        1 [11097750, 11097868]      - |       102 ENSE00003482788
##    ...      ...                  ...    ... .       ...             ...
##   [14]        1 [11106948, 11107176]      - |       111 ENSE00003467404
##   [15]        1 [11106948, 11107176]      - |       112 ENSE00003489217
##   [16]        1 [11107260, 11107280]      - |       113 ENSE00001833377
##   [17]        1 [11107260, 11107284]      - |       114 ENSE00001472289
##   [18]        1 [11107260, 11107290]      - |       115 ENSE00001881401
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The `rowRanges` also contains metadata about the construction of the gene model in the `metadata` slot. Here we use a helpful R function, `str`, to display the metadata compactly:

```
str(metadata(rowRanges(se)))
```

File failed to load: /extensions/MathZoom.js

```
## List of 1
##  $ genomeInfo:List of 15
##   ..$ Db type                                 : chr "TxDb"
##   ..$ Supporting package                      : chr "GenomicFeatures"
##   ..$ Data source                             : chr "/Library/Frameworks/R.framework/
Versions/3.3/Resources/library/airway/extdata/Homo_sapiens.GRCh37.75_subset.gtf"
##   ..$ Organism                                : chr NA
##   ..$ Taxonomy ID                             : chr NA
##   ..$ miRBase build ID                        : chr NA
##   ..$ Genome                                  : chr NA
##   ..$ transcript_nrow                         : chr "65"
##   ..$ exon_nrow                               : chr "279"
##   ..$ cds_nrow                                : chr "158"
##   ..$ Db created by                           : chr "GenomicFeatures package from Bio
conductor"
##   ..$ Creation time                           : chr "2016-07-11 15:51:17 +0200 (Mon,
 11 Jul 2016)"
##   ..$ GenomicFeatures version at creation time: chr "1.24.2"
##   ..$ RSQLite version at creation time        : chr "1.0.0"
##   ..$ DBSCHEMAVERSION                         : chr "1.1"
```

The `colData` stores the metadata about the samples:

```
colData(se)
```

```
## DataFrame with 8 rows and 0 columns
```

The `colData` slot is so far empty!

Because we used a column of `sampleTable` to produce the `bamfiles` vector, we know the columns of `se` are in the same order as the rows of `sampleTable`. Take a moment to convince yourself this is true:

```
colnames(se)
```

```
## [1] "SRR1039508_subset.bam" "SRR1039509_subset.bam" "SRR1039512_subset.bam" "SRR10395
13_subset.bam"
## [5] "SRR1039516_subset.bam" "SRR1039517_subset.bam" "SRR1039520_subset.bam" "SRR10395
21_subset.bam"
```

```
bamfiles
```

```
## BamFileList of length 8
## names(8): SRR1039508_subset.bam SRR1039509_subset.bam ... SRR1039521_subset.bam
```

```
sampleTable$Run
```

File failed to load: /extensions/MathZoom.js

```
## [1] SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516 SRR1039517 SRR1039520 SRR1
039521
## 8 Levels: SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516 SRR1039517 ... SRR10
39521
```

We can assign the `sampleTable` as the `colData` of the summarized experiment, by converting it into a *DataFrame* and using the assignment function:

```
colData(se) <- DataFrame(sampleTable)
colData(se)
```

```
## DataFrame with 8 rows and 9 columns
##            SampleName     cell      dex    albut        Run avgLength Experiment    S
ample
##              <factor> <factor> <factor> <factor>   <factor> <integer>   <factor>  <fa
ctor>
## SRR1039508 GSM1275862   N61311    untrt    untrt SRR1039508       126  SRX384345 SRS5
08568
## SRR1039509 GSM1275863   N61311      trt    untrt SRR1039509       126  SRX384346 SRS5
08567
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126  SRX384349 SRS5
08571
## SRR1039513 GSM1275867  N052611      trt    untrt SRR1039513        87  SRX384350 SRS5
08572
## SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120  SRX384353 SRS5
08575
## SRR1039517 GSM1275871  N080611      trt    untrt SRR1039517       126  SRX384354 SRS5
08576
## SRR1039520 GSM1275874  N061011    untrt    untrt SRR1039520       101  SRX384357 SRS5
08579
## SRR1039521 GSM1275875  N061011      trt    untrt SRR1039521        98  SRX384358 SRS5
08580
##               BioSample
##                <factor>
## SRR1039508 SAMN02422669
## SRR1039509 SAMN02422675
## SRR1039512 SAMN02422678
## SRR1039513 SAMN02422670
## SRR1039516 SAMN02422682
## SRR1039517 SAMN02422673
## SRR1039520 SAMN02422683
## SRR1039521 SAMN02422677
```

We are now finished exploring the parts of the *SummarizedExperiment*.

Alternative - Counting with featureCounts in Rsubread

File failed to load: /extensions/MathZoom.js

Another option for counting reads or fragments within R/Bioconductor is the *Rsubread (http://bioconductor.org/packages/Rsubread)* package which contains the *featureCounts* function (Liao, Smyth, and Shi 2014). This is very simple to use and very fast, and returns the count matrix as part of the result. See `?featureCounts` for more information on its usage, including how to sort the BAM files for fastest counting.

When you run *featureCounts* you will see a large logo printed to your screen as well as other information displayed live as the software is counting.

```
library("Rsubread")
fc <- featureCounts(files=filenames,
                    annot.ext=gtffile,
                    isGTFAnnotationFile=TRUE,
                    isPairedEnd=TRUE)
```

```
##
##         ==========     _____ _    _ ____  _____  ______          _____
##         =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
##           =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
##             ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
##               ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
##         ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
##        Rsubread 1.22.2
##
## //========================== featureCounts setting ===========================\\
## ||                                                                            ||
## ||             Input files : 8 BAM files                                      ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                           P /Library/Frameworks/R.framework/Versions/3 ... ||
## ||                                                                            ||
## ||             Output file : ./.Rsubread_featureCounts_pid14300               ||
## ||                 Summary : ./.Rsubread_featureCounts_pid14300.summary       ||
## ||              Annotation : /Library/Frameworks/R.framework/Versions/3.3 ... ||
## ||                                                                            ||
## ||                 Threads : 1                                                ||
## ||                   Level : meta-feature level                               ||
## ||              Paired-end : yes                                              ||
## ||         Strand specific : no                                               ||
## ||      Multimapping reads : not counted                                      ||
## || Multi-overlapping reads : not counted                                      ||
## ||                                                                            ||
## ||          Chimeric reads : counted                                          ||
## ||        Both ends mapped : not required                                     ||
## ||                                                                            ||
## \\===================== http://subread.sourceforge.net/ ======================//
##
## //================================= Running ==================================\\
## ||                                                                            ||
## || Load annotation file /Library/Frameworks/R.framework/Versions/3.3/Reso ... ||
## ||    Features : 406                                                          ||
## ||    Meta-features : 20                                                      ||
## ||    Chromosomes/contigs : 1                                                 ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 7142                                                  ||
## ||    Successfully assigned fragments : 6649 (93.1%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 7200                                                  ||
## ||    Successfully assigned fragments : 6712 (93.2%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 8536                                                  ||
## ||    Successfully assigned fragments : 7910 (92.7%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 7544                                                  ||
## ||    Successfully assigned fragments : 7044 (93.4%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 8818                                                  ||
## ||    Successfully assigned fragments : 8261 (93.7%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 11850                                                 ||
## ||    Successfully assigned fragments : 11148 (94.1%)                         ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 5877                                                  ||
## ||    Successfully assigned fragments : 5415 (92.1%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Process BAM file /Library/Frameworks/R.framework/Versions/3.3/Resource ... ||
## ||    Paired-end reads are included.                                          ||
## ||    Assign fragments (read pairs) to features...                            ||
## ||                                                                            ||
## ||    WARNING: reads from the same pair were found not adjacent to each       ||
## ||             other in the input (due to read sorting by location or         ||
## ||             reporting of multi-mapping read pairs).                        ||
## ||                                                                            ||
## ||    Read re-ordering is performed.                                          ||
## ||                                                                            ||
## ||    Total fragments : 10208                                                 ||
## ||    Successfully assigned fragments : 9538 (93.4%)                          ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## ||                         Read assignment finished.                          ||
## ||                                                                            ||
## \\===================== http://subread.sourceforge.net/ ======================//
```

File failed to load: /extensions/MathZoom.js

```
colnames(fc$counts) <- sampleTable$Run
head(fc$counts)
```

```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516 SRR1039517 SRR
1039520
## ENSG00000175262          0          0          4          1          0          0
      1
## ENSG00000215785          0          0          1          0          0          0
      0
## ENSG00000264181          0          0          0          0          0          0
      0
## ENSG00000207213          0          0          0          0          0          0
      0
## ENSG00000120948       2673       2031       3263       1570       3446       3615
   2171
## ENSG00000009724         38         28         66         24         42         41
     47
##                 SRR1039521
## ENSG00000175262          0
## ENSG00000215785          0
## ENSG00000264181          0
## ENSG00000207213          0
## ENSG00000120948       1949
## ENSG00000009724         36
```

Branching point

At this point, we have counted the fragments which overlap the genes in the gene model we specified. This is a branching point where we could use a variety of Bioconductor packages for exploration and differential expression of the count matrix, including *edgeR (http://bioconductor.org/packages/edgeR)* (M. D. Robinson, McCarthy, and Smyth 2009), *limma (http://bioconductor.org/packages/limma)* with the voom method (Law et al. 2014), *DSS (http://bioconductor.org/packages/DSS)* (H. Wu, Wang, and Wu 2013), *EBSeq (http://bioconductor.org/packages/EBSeq)* (Leng et al. 2013) and *BaySeq (http://bioconductor.org/packages/BaySeq)* (Hardcastle and Kelly 2010).

We will continue, using *DESeq2 (http://bioconductor.org/packages/DESeq2)* (Love, Huber, and Anders 2014) and *edgeR (http://bioconductor.org/packages/edgeR)* (M. D. Robinson, McCarthy, and Smyth 2009). Each of these packages has a specific class of object used to store the summarization of the RNA-seq experiment, and the intermediate quantities that are calculated during the statistical analysis of the data. *DESeq2* uses a *DESeqDataSet* and *edgeR* uses a *DGEList*.

The *DESeqDataSet*, sample information, and the design formula

Bioconductor software packages often define and use a custom class for storing data that makes sure that all the needed data slots are consistently provided and fulfill the requirements. In addition, Bioconductor has general data classes (such as the *SummarizedExperiment*) that can be used to move data between packages. Additionally, the core Bioconductor classes provide useful functionality: for example, subsetting or reordering the rows or columns of a *SummarizedExperiment* automatically subsets or reorders the associated *rowRanges* and *colData*, which can help to prevent accidental sample swaps that would otherwise lead to spurious results. With *SummarizedExperiment* this is all taken care of behind the scenes.

In *DESeq2*, the custom class is called *DESeqDataSet*. It is built on top of the *SummarizedExperiment* class, and it is easy to convert *SummarizedExperiment* objects into *DESeqDataSet* objects, which we show below. One of the two main differences is that the `assay` slot is instead accessed using the *counts* accessor function, and the *DESeqDataSet* class enforces that the values in this matrix are non-negative integers.

A second difference is that the *DESeqDataSet* has an associated *design formula*. The experimental design is specified at the beginning of the analysis, as it will inform many of the *DESeq2* functions how to treat the samples in the analysis (one exception is the size factor estimation, i.e., the adjustment for differing library sizes, which does not depend on the design formula). The design formula tells which columns in the sample information table (`colData`) specify the experimental design and how these factors should be used in the analysis.

The simplest design formula for differential expression would be `~ condition` , where `condition` is a column in `colData(dds)` that specifies which of two (or more groups) the samples belong to.

However, let's remind ourselves of the experimental design of the experiment:

```
colData(se)
```

```
## DataFrame with 8 rows and 9 columns
##            SampleName     cell      dex    albut        Run avgLength Experiment    S
ample
##              <factor> <factor> <factor> <factor>   <factor> <integer>   <factor>  <fa
ctor>
## SRR1039508 GSM1275862   N61311    untrt    untrt SRR1039508       126  SRX384345 SRS5
08568
## SRR1039509 GSM1275863   N61311      trt    untrt SRR1039509       126  SRX384346 SRS5
08567
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126  SRX384349 SRS5
08571
## SRR1039513 GSM1275867  N052611      trt    untrt SRR1039513        87  SRX384350 SRS5
08572
## SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120  SRX384353 SRS5
08575
## SRR1039517 GSM1275871  N080611      trt    untrt SRR1039517       126  SRX384354 SRS5
08576
## SRR1039520 GSM1275874  N061011    untrt    untrt SRR1039520       101  SRX384357 SRS5
08579
## SRR1039521 GSM1275875  N061011      trt    untrt SRR1039521        98  SRX384358 SRS5
08580
##               BioSample
##                <factor>
## SRR1039508 SAMN02422669
## SRR1039509 SAMN02422675
## SRR1039512 SAMN02422678
## SRR1039513 SAMN02422670
## SRR1039516 SAMN02422682
## SRR1039517 SAMN02422673
## SRR1039520 SAMN02422683
## SRR1039521 SAMN02422677
```

File failed to load: /extensions/MathZoom.js

We have treated and untreated samples (as indicated by `dex`):

```
se$dex
```

```
## [1] untrt trt   untrt trt   untrt trt   untrt trt  
## Levels: trt untrt
```

We also have four different cell lines:

```
se$cell
```

```
## [1] N61311  N61311  N052611 N052611 N080611 N080611 N061011 N061011
## Levels: N052611 N061011 N080611 N61311
```

We want to compare the differences in gene expression that can be associated with dexamethasone treatment, but we also want to control for differences across the four cell lines. The design which accomplishes this is to write `~ cell + dex`. By including `cell`, terms will be added to the model which account for differences across cell, and by adding `dex` we get a single term which explains the differences across treated and untreated samples.

Note: it will be helpful for us if the first level of a factor be the reference level (e.g. control, or untreated samples). The reason is that, by specifying this, functions further in the pipeline can be used and will give comparisons such as, treatment vs control, without needing to specify additional arguments.

We can *relevel* the `dex` factor like so:

```
se$dex <- relevel(se$dex, "untrt")
se$dex
```

```
## [1] untrt trt   untrt trt   untrt trt   untrt trt  
## Levels: untrt trt
```

It is not important for us to *relevel* the `cell` variable, nor is there a clear reference level for cell line.

For running *DESeq2* or *edgeR* models, you can use R's formula notation to express any fixed-effects experimental design. Note that these packages use the same formula notation as, for instance, the *lm* function of base R. If the research aim is to determine for which genes the effect of treatment is different across groups, then interaction terms can be included and tested using a design such as `~ group + treatment + group:treatment`. See the vignettes of *DESeq2* and *edgeR* for more examples.

In the following sections, we will demonstrate the construction of the *DESeqDataSet* from two starting points:

- from a *SummarizedExperiment* object
- from a count matrix and a sample information table

Starting from *SummarizedExperiment*

We now use R's *data* command to load a prepared *SummarizedExperiment* that was generated from the publicly available sequencing data files associated with Himes et al. (2014), described above. The steps we used to produce this object were equivalent to those you worked through in the previous sections, except that we used all

File failed to load: /extensions/MathZoom.js

the reads and all the genes. For more details on the exact steps used to create this object, type `vignette("airway")` into your R session.

```
data("airway")
se <- airway
```

Pre-filtering rows with very small counts

This *SummarizedExperiment* contains many rows (genes) with zero or very small counts. In order to streamline the workflow, which uses multiple packages, we will remove those genes which have a total count of less than 5:

```
se <- se[ rowSums(assay(se)) >= 5, ]
```

Again, we want to specify that `untrt` is the reference level for the dex variable:

```
se$dex <- relevel(se$dex, "untrt")
se$dex
```

```
## [1] untrt trt   untrt trt   untrt trt   untrt trt
## Levels: untrt trt
```

Supposing we have constructed a *SummarizedExperiment* using one of the methods described in the previous section, we now need to make sure that the object contains all the necessary information about the samples, i.e., a table with metadata on the count matrix's columns stored in the `colData` slot:

```
colData(se)
```

File failed to load: /extensions/MathZoom.js

```
## DataFrame with 8 rows and 9 columns
##            SampleName     cell      dex    albut        Run avgLength Experiment    S
ample
##              <factor> <factor> <factor> <factor>   <factor> <integer>   <factor>  <fa
ctor>
## SRR1039508 GSM1275862   N61311    untrt    untrt SRR1039508       126  SRX384345 SRS5
08568
## SRR1039509 GSM1275863   N61311      trt    untrt SRR1039509       126  SRX384346 SRS5
08567
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126  SRX384349 SRS5
08571
## SRR1039513 GSM1275867  N052611      trt    untrt SRR1039513        87  SRX384350 SRS5
08572
## SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120  SRX384353 SRS5
08575
## SRR1039517 GSM1275871  N080611      trt    untrt SRR1039517       126  SRX384354 SRS5
08576
## SRR1039520 GSM1275874  N061011    untrt    untrt SRR1039520       101  SRX384357 SRS5
08579
## SRR1039521 GSM1275875  N061011      trt    untrt SRR1039521        98  SRX384358 SRS5
08580
##               BioSample
##                <factor>
## SRR1039508 SAMN02422669
## SRR1039509 SAMN02422675
## SRR1039512 SAMN02422678
## SRR1039513 SAMN02422670
## SRR1039516 SAMN02422682
## SRR1039517 SAMN02422673
## SRR1039520 SAMN02422683
## SRR1039521 SAMN02422677
```

Here we see that this object already contains an informative `colData` slot – because we have already prepared it for you, as described in the *airway (http://bioconductor.org/packages/airway)* vignette. However, when you work with your own data, you will have to add the pertinent sample / phenotypic information for the experiment at this stage. We highly recommend keeping this information in a comma-separated value (CSV) or tab-separated value (TSV) file, which can be exported from an Excel spreadsheet, and the assign this to the `colData` slot, making sure that the rows correspond to the columns of the *SummarizedExperiment*. We made sure of this correspondence earlier by specifying the BAM files using a column of the sample table.

Once we have our fully annotated *SummarizedExperiment* object, we can construct a *DESeqDataSet* object from it that will then form the starting point of the analysis. We add an appropriate design for the analysis:

```
library("DESeq2")
dds <- DESeqDataSet(se, design = ~ cell + dex)
```

Starting from count matrices

In this section, we will show how to build an *DESeqDataSet* supposing we only have a count matrix and a table of sample information.

File failed to load: /extensions/MathZoom.js

Note: if you have prepared a *SummarizedExperiment* you should skip this section. While the previous section would be used to construct a *DESeqDataSet* from a *SummarizedExperiment*, here we first extract the individual object (count matrix and sample info) from the *SummarizedExperiment* in order to build it back up into a new object – only for demonstration purposes. In practice, the count matrix would either be read in from a file or perhaps generated by an R function like *featureCounts* from the *Rsubread (http://bioconductor.org/packages/Rsubread)* package (Liao, Smyth, and Shi 2014).

The information in a *SummarizedExperiment* object can be accessed with accessor functions. For example, to see the actual data, i.e., here, the fragment counts, we use the *assay* function. (The *head* function restricts the output to the first few lines.)

```
countdata <- assay(se)
head(countdata, 3)
```

```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516 SRR1039517 SRR
1039520
## ENSG00000000003        679        448        873        408       1138       1047
    770
## ENSG00000000419        467        515        621        365        587        799
    417
## ENSG00000000457        260        211        263        164        245        331
    233
##                 SRR1039521
## ENSG00000000003        572
## ENSG00000000419        508
## ENSG00000000457        229
```

In this count matrix, each row represents an Ensembl gene, each column a sequenced RNA library, and the values give the raw numbers of fragments that were uniquely assigned to the respective gene in each library. We also have information on each of the samples (the columns of the count matrix). If you've counted reads with some other software, it is very important to check that the columns of the count matrix correspond to the rows of the sample information table.

```
coldata <- colData(se)
```

We now have all the ingredients to prepare our data object in a form that is suitable for analysis, namely:

- `countdata` : a table with the fragment counts
- `coldata` : a table with information about the samples

To now construct the *DESeqDataSet* object from the matrix of counts and the sample information table, we use:

```
ddsMat <- DESeqDataSetFromMatrix(countData = countdata,
                                 colData = coldata,
                                 design = ~ cell + dex)
```

Creating a DGEList for use with edgeR

As mentioned above, the edgeR package uses another type of data container, namely a DGEList object. It is just as easy to create a *DGEList* object using the count matrix and information about samples. We can additionally add information about the genes:

```
library("edgeR")
genetable <- data.frame(gene.id=rownames(se))
y <- DGEList(counts=countdata,
             samples=coldata,
             genes=genetable)
names(y)
```

```
## [1] "counts"  "samples" "genes"
```

Just like the *SummarizedExperiment* and the *DESeqDataSet* the *DGEList* contains all the information we need to know: the count matrix, information about the samples (columns of the count matrix), and information about the genes (rows of the count matrix).

Exploratory analysis and visualization

There are two separate paths in this workflow:

1. *visually exploring* sample relationships, in which we will discuss transformation of the counts for computing distances or making plots
2. *statistical testing* for differences attributable to treatment, controlling for cell line effects

Importantly, the statistical testing methods rely on original count data (not scaled or transformed) for calculating the precision of measurements. However, for visualization and exploratory analysis, transformed counts are typically more suitable. Thus, it is critical to separate the two workflows and use the appropriate input data for each of them.

Transformations

Many common statistical methods for exploratory analysis of multidimensional data, for example clustering and *principal components analysis* (PCA), work best for data that generally has the same range of variance at different ranges of the mean values. When the expected amount of variance is approximately the same across different mean values, the data is said to be *homoskedastic*. For RNA-seq raw counts, however, the variance grows with the mean. For example, if one performs PCA directly on a matrix of size-factor-normalized read counts, the result typically depends only on the few most strongly expressed genes because they show the largest absolute differences between samples. A simple and often used strategy to avoid this is to take the logarithm of the normalized count values plus a small pseudocount; however, now the genes with the very lowest counts will tend to dominate the results because, due to the strong Poisson noise inherent to small count values, and the fact that the logarithm amplifies differences for the smallest values, these low count genes will show the strongest relative differences between samples.

As a solution, *DESeq2* offers transformations for count data that stabilize the variance across the mean: the *regularize logarithm* (rlog) and the *variance stabilizing transformation* (VST). These have slightly different implementations, discussed a bit in the *DESeq2* paper and in the vignette, but a similar goal of stablizing the variance across the range of values. Both produce log2-like values for high counts. Here we will use the variance stabilizing transformation implemented with the `vst` function:

```
vsd <- vst(dds)
```

File failed to load: /extensions/MathZoom.js

This returns a *DESeqTransform* object

```
class(vsd)
```

```
## [1] "DESeqTransform"
## attr(,"package")
## [1] "DESeq2"
```

…which contains all the column metadata that was attached to the *DESeqDataSet*:

```
head(colData(vsd),3)
```

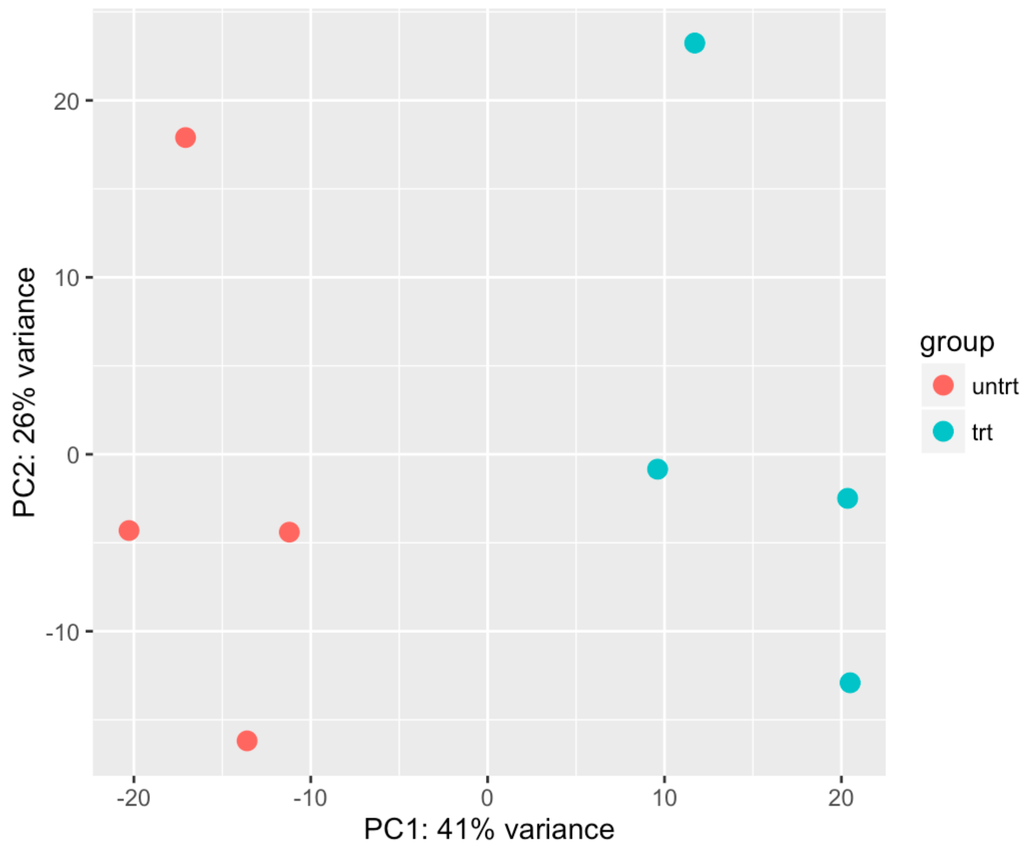
```
## DataFrame with 3 rows and 10 columns
##             SampleName     cell      dex    albut        Run avgLength Experiment    S
ample
##               <factor> <factor> <factor> <factor>   <factor> <integer>   <factor>  <fa
ctor>
## SRR1039508 GSM1275862   N61311    untrt    untrt SRR1039508       126  SRX384345 SRS5
08568
## SRR1039509 GSM1275863   N61311      trt    untrt SRR1039509       126  SRX384346 SRS5
08567
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126  SRX384349 SRS5
08571
##               BioSample sizeFactor
##                <factor>  <numeric>
## SRR1039508 SAMN02422669  1.0236476
## SRR1039509 SAMN02422675  0.8961667
## SRR1039512 SAMN02422678  1.1794861
```

PCA plot

Another way to visualize sample-to-sample distances is a principal components analysis (PCA). In this ordination method, the data points (here, the samples) are projected onto the 2D plane such that they spread out in the two directions that explain most of the differences (Figure below). The x-axis is the direction that separates the data points the most. The values of the samples in this direction are written *PC1*. The y-axis is a direction (it must be *orthogonal* to the first direction) that separates the data the second most. The values of the samples in this direction are written *PC2*. The percent of the total variance that is contained in the direction is printed in the axis label. Note that these percentages do not add to 100%, because there are more dimensions that contain the remaining variance (although each of these remaining dimensions will explain less than the two that we see).

```
plotPCA(vsd, "dex")
```

File failed to load: /extensions/MathZoom.js

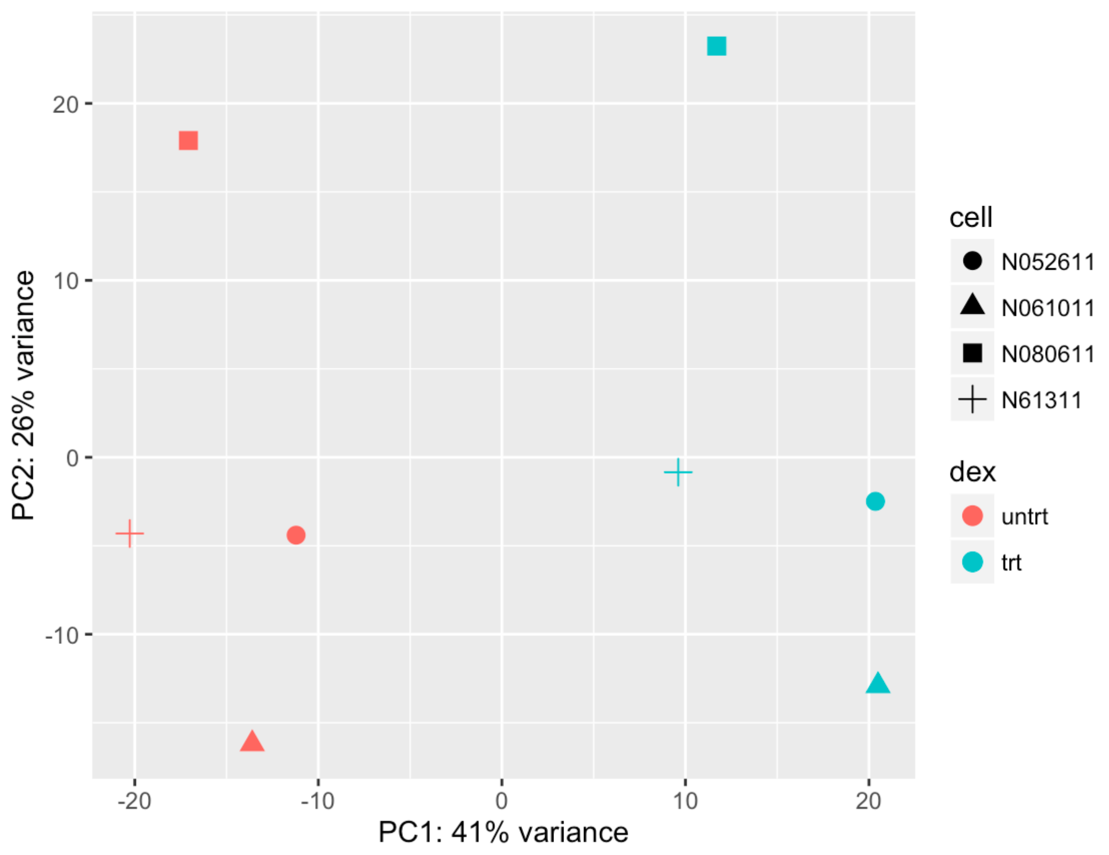


We can also build the PCA plot from scratch using the *ggplot2 (http://cran.fhcrc.org/web/packages/ggplot2/index.html)* package (Wickham 2009). This is done by asking the *plotPCA* function to return the data used for plotting rather than building the plot. See the *ggplot2* documentation (http://docs.ggplot2.org/current/) for more details on using *ggplot*.

```
data <- plotPCA(vsd, intgroup = c( "dex", "cell"), returnData=TRUE)
percentVar <- round(100 * attr(data, "percentVar"))
```

We can then use this data to build up a second plot in a Figure below, specifying that the color of the points should reflect dexamethasone treatment and the shape should reflect the cell line.

```
library("ggplot2")
ggplot(data, aes(PC1, PC2, color=dex, shape=cell)) + geom_point(size=3) +
  xlab(paste0("PC1: ",percentVar[1],"% variance")) +
  ylab(paste0("PC2: ",percentVar[2],"% variance"))
```



Here we specify cell line (plotting symbol) and dexamethasone treatment (color).

From the PCA plot, we see that the differences between cells (the different plotting shapes) are considerable, though not stronger than the differences due to treatment with dexamethasone (red vs blue color). This shows why it will be important to account for the cell line effect in differential testing by using a paired design ("paired", because each dex treated sample is paired with one untreated sample from the *same* cell line). We are already set up for this design by assigning the formula `~ cell + dex` earlier.

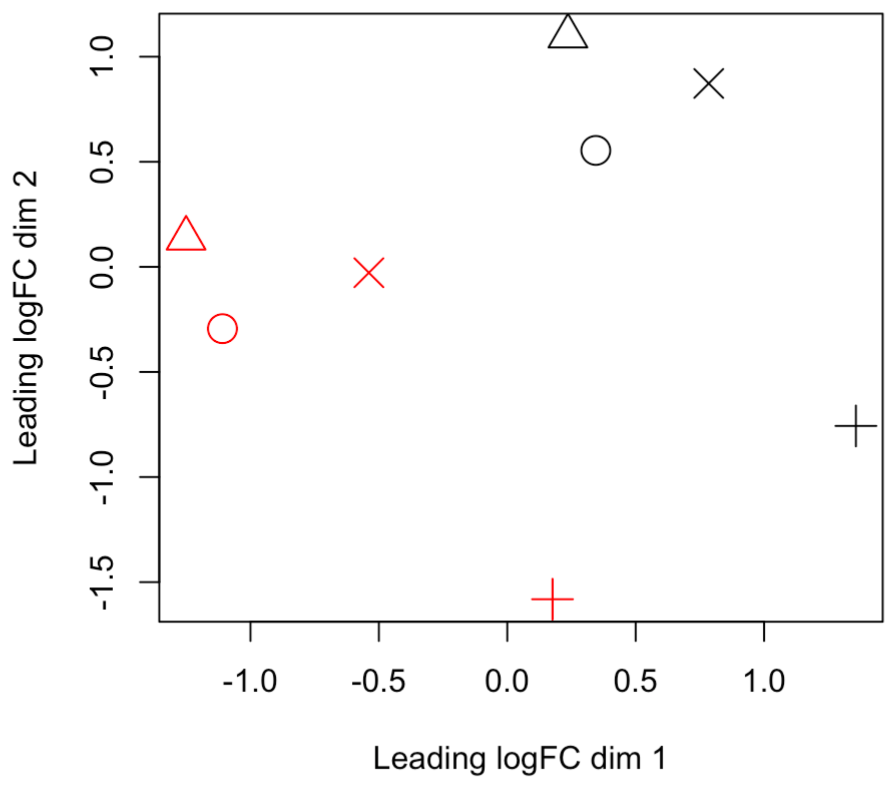
MDS plot

Another way to reduce dimensionality, which is in many ways similar to PCA, is *multidimensional scaling* (MDS). For MDS, we first have to calculate all pairwise distances between our objects (samples in this case), and then create a (typically) two-dimensional representation where these pre-calculated distances are represented as accurately as possible. This means that depending on how the pairwise sample distances are defined, the two-dimensional plot can be very different, and it is important to choose a distance that is suitable for the type of data at hand.

edgeR contains a function `plotMDS`, which operates on a `DGEList` object and generates a two-dimensional MDS representation of the samples. The default distance between two samples can be interpreted as the "typical" log fold change between the two samples, for the genes that are most different between them (by default, the top 500 genes, but this can be modified as shown below). We generate an MDS plot from the `DGEList` object `y`, coloring by the treatment and using different plot symbols for different cell lines.

```
y <- calcNormFactors(y)
plotMDS(y, top = 1000, labels = NULL, col = as.numeric(y$samples$dex),
        pch = as.numeric(y$samples$cell), cex = 2)
```

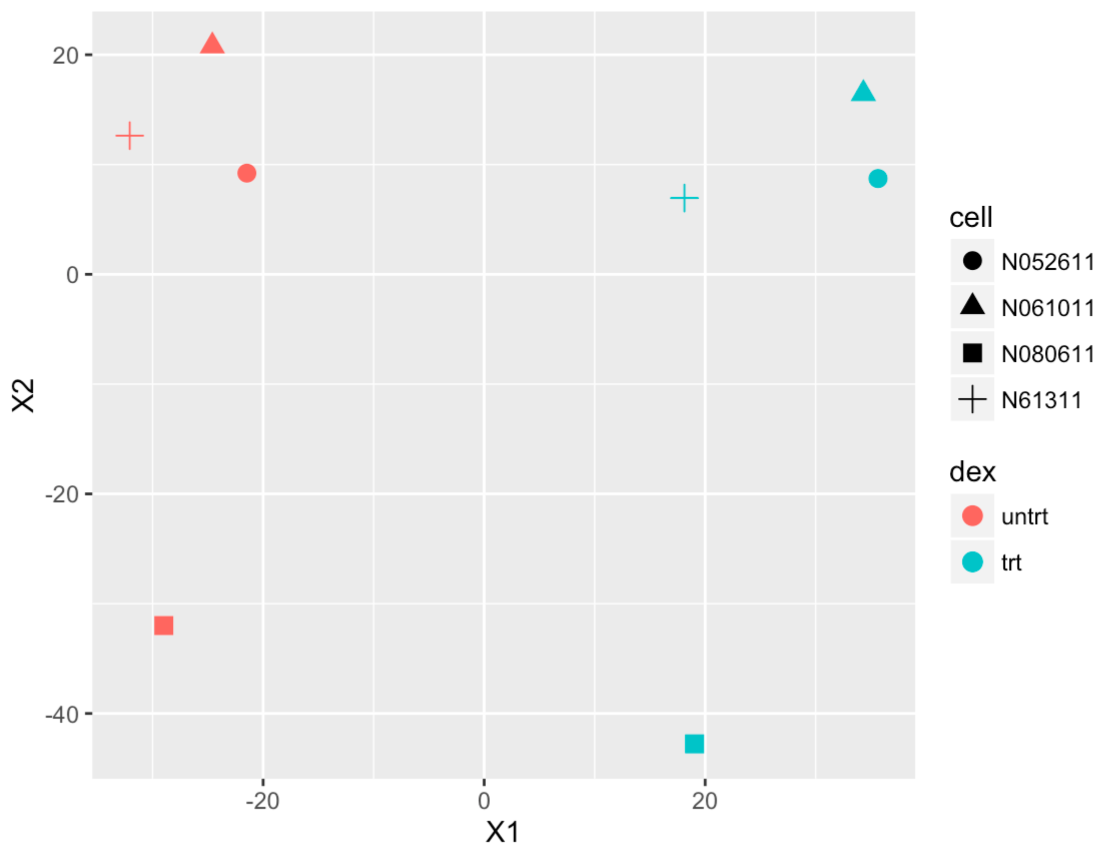
File failed to load: /extensions/MathZoom.js



We can also perform MDS on manually calculated distances, using the R function `cmdscale`. Below we show how to do this starting from the VST counts, using Euclidean distance.

```
sampleDists <- dist(t(assay(vsd)))
sampleDistMatrix <- as.matrix( sampleDists )
mdsData <- data.frame(cmdscale(sampleDistMatrix))
mds <- cbind(mdsData, as.data.frame(colData(vsd)))
ggplot(mds, aes(X1,X2,color=dex,shape=cell)) + geom_point(size=3)
```

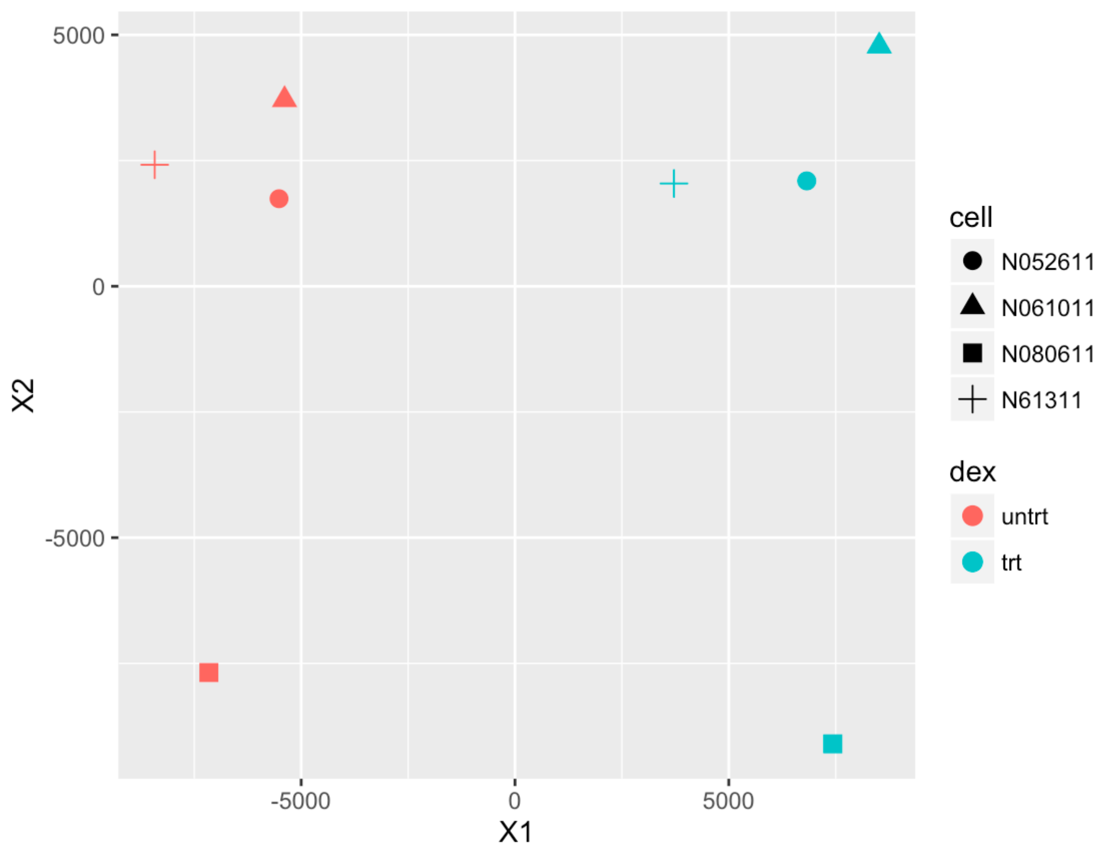
File failed to load: /extensions/MathZoom.js



In a Figure below we show the same plot for the *PoissonDistance*:

```
library("PoiClaClu")
poisd <- PoissonDistance(t(counts(dds)))
samplePoisDistMatrix <- as.matrix( poisd$dd )
mdsPoisData <- data.frame(cmdscale(samplePoisDistMatrix))
mdsPois <- cbind(mdsPoisData, as.data.frame(colData(dds)))
ggplot(mdsPois, aes(X1,X2,color=dex,shape=cell)) + geom_point(size=3)
```

File failed to load: /extensions/MathZoom.js



Differential expression analysis

Performing differential expression testing with DESeq2

As we have already specified an experimental design when we created the *DESeqDataSet*, we can run the differential expression pipeline on the raw counts with a single call to the function *DESeq*:

```
dds <- DESeq(dds)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
## fitting model and testing
```

File failed to load: /extensions/MathZoom.js

This function will print out a message for the various steps it performs. These are described in more detail in the manual page for *DESeq*, which can be accessed by typing `?DESeq`. Briefly these are: the estimation of size factors (controlling for differences in the sequencing depth of the samples), the estimation of dispersion values for each gene, and fitting a generalized linear model.

A *DESeqDataSet* is returned that contains all the fitted parameters within it, and the following section describes how to extract out results tables of interest from this object.

We will show in a following section how to perform differential testing using edgeR.

Building the results table

Calling *results* without any arguments will extract the estimated log2 fold changes and *p* values for the last variable in the design formula. If there are more than 2 levels for this variable, *results* will extract the results table for a comparison of the last level over the first level. This comparison is printed at the top of the output: `dex trt vs untrt`.

```
res <- results(dds)
```

As `res` is a *DataFrame* object, it carries metadata with information on the meaning of the columns:

```
mcols(res, use.names=TRUE)
```

```
## DataFrame with 6 rows and 2 columns
##                        type                               description
##                 <character>                               <character>
## baseMean       intermediate mean of normalized counts for all samples
## log2FoldChange      results  log2 fold change (MAP): dex trt vs untrt
## lfcSE               results          standard error: dex trt vs untrt
## stat                results          Wald statistic: dex trt vs untrt
## pvalue              results       Wald test p-value: dex trt vs untrt
## padj                results                      BH adjusted p-values
```

The first column, `baseMean`, is a just the average of the normalized count values, dividing by size factors, taken over all samples in the *DESeqDataSet*. The remaining four columns refer to a specific contrast, namely the comparison of the `trt` level over the `untrt` level for the factor variable `dex`. We will find out below how to obtain other contrasts.

The column `log2FoldChange` is the effect size estimate. It tells us how much the gene's expression seems to have changed due to treatment with dexamethasone in comparison to untreated samples. This value is reported on a logarithmic scale to base 2: for example, a log2 fold change of 1.5 means that the gene's expression is increased by a multiplicative factor of \(2^{1.5} \approx 2.82\).

Of course, this estimate has an uncertainty associated with it, which is available in the column `lfcSE`, the standard error estimate for the log2 fold change estimate. We can also express the uncertainty of a particular effect size estimate as the result of a statistical test. The purpose of a test for differential expression is to test whether the data provides sufficient evidence to conclude that this value is really different from zero. *DESeq2* performs for each gene a *hypothesis test* to see whether evidence is sufficient to decide against the *null hypothesis* that there is zero effect of the treatment on the gene and that the observed difference between treatment and control was merely caused by experimental variability (i.e., the type of variability that you can expect between different samples in the same treatment group). As usual in statistics, the result of this test is

File failed to load: /extensions/MathZoom.js

reported as a *p* value, and it is found in the column `pvalue`. Remember that a *p* value indicates the probability that a fold change as strong as the observed one, or even stronger, would be seen under the situation described by the null hypothesis.

We can also summarize the results with the following line of code, which reports some additional information, that will be covered in later sections.

```
summary(res)
```

```
##
## out of 25133 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)     : 2630, 10%
## LFC < 0 (down)   : 2216, 8.8%
## outliers [1]     : 0, 0%
## low counts [2]   : 7309, 29%
## (mean count < 5)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Note that there are many genes with differential expression due to dexamethasone treatment at the FDR level of 10%. This makes sense, as the smooth muscle cells of the airway are known to react to glucocorticoid steroids. However, there are two ways to be more strict about which set of genes are considered significant:

- lower the false discovery rate threshold (the threshold on `padj` in the results table)
- raise the log2 fold change threshold from 0 using the `lfcThreshold` argument of *results*

If we lower the false discovery rate threshold, we should also tell this value to `results()`, so that the function will use an alternative threshold for the optimal independent filtering step:

```
res.05 <- results(dds, alpha=.05)
table(res.05$padj < .05)
```

```
##
## FALSE  TRUE
## 13313  4023
```

If we want to raise the log2 fold change threshold, so that we test for genes that show more substantial changes due to treatment, we simply supply a value on the log2 scale. For example, by specifying `lfcThreshold=1`, we test for genes that show significant effects of treatment on gene counts more than doubling or less than halving, because \(2^1 = 2\).

```
resLFC1 <- results(dds, lfcThreshold=1)
table(resLFC1$padj < 0.1)
```

```
##
## FALSE  TRUE
## 18602   196
```

File failed to load: /extensions/MathZoom.js

Sometimes a subset of the *p* values in `res` will be `NA` (“not available”). This is *DESeq*’s way of reporting that all counts for this gene were zero, and hence no test was applied. In addition, *p* values can be assigned `NA` if the gene was excluded from analysis because it contained an extreme count outlier. For more information, see the outlier detection section of the *DESeq2* vignette.

Performing differential expression testing with edgeR

Next we will show how to perform differential expression analysis with *edgeR*. Recall that we have a `DGEList` `y`, containing three objects:

```
names(y)
```

```
## [1] "counts"  "samples" "genes"
```

We first define a design matrix, using the same formula syntax as for *DESeq2* above.

```
design <- model.matrix(~ cell + dex, y$samples)
```

Then, we calculate normalization factors and estimate the dispersion for each gene. Note that we need to specify the design in the dispersion calculation.

```
y <- calcNormFactors(y)
y <- estimateDisp(y, design)
```

Finally, we fit the generalized linear model and perform the test. In the `glmLRT` function, we indicate which coefficient (which column in the design matrix) that we would like to test for. It is possible to test more general contrasts as well, and the user guide contains many examples on how to do this. The `topTags` function extracts the top-ranked genes. You can indicate the adjusted p-value cutoff, and/or the number of genes to keep.

```
fit <- glmFit(y, design)
lrt <- glmLRT(fit, coef=ncol(design))
tt <- topTags(lrt, n=nrow(y), p.value=0.1)
tt10 <- topTags(lrt) # just the top 10 by default
tt10
```

```
## Coefficient:  dextrt
##               gene.id     logFC   logCPM        LR        PValue           FDR
## 3332  ENSG00000109906  7.150141 4.148709 1291.8953 6.523720e-283 1.639607e-278
## 8361  ENSG00000152583  4.559193 5.533094  900.0568 9.538511e-198 1.198657e-193
## 10178 ENSG00000165995  3.280578 4.505772  741.6181 2.666490e-163 2.233897e-159
## 9719  ENSG00000163884  4.467109 4.689483  718.1782 3.332858e-158 2.094118e-154
## 5257  ENSG00000127954  5.201346 3.649772  652.7413 5.661641e-144 2.845881e-140
## 13939 ENSG00000189221  3.341865 6.769186  650.1580 2.064214e-143 8.646650e-140
## 2274  ENSG00000101347  3.758928 9.208366  646.4330 1.333112e-142 4.786442e-139
## 9432  ENSG00000162692 -3.670006 4.596955  640.0100 3.324746e-141 1.044511e-137
## 11274 ENSG00000171819  5.668732 3.501178  612.3666 3.419855e-135 9.550136e-132
## 4475  ENSG00000120129  2.939177 7.310512  593.7476 3.834917e-131 9.638296e-128
```

File failed to load: /extensions/MathZoom.js

We can compare to see how the results from the two software overlap:

```
tt.all <- topTags(lrt, n=nrow(y), sort.by="none")
table(DESeq2=res$padj < 0.1, edgeR=tt.all$table$FDR < 0.1)
```

```
##        edgeR
## DESeq2  FALSE  TRUE
##   FALSE 11528  1450
##   TRUE    106  4740
```

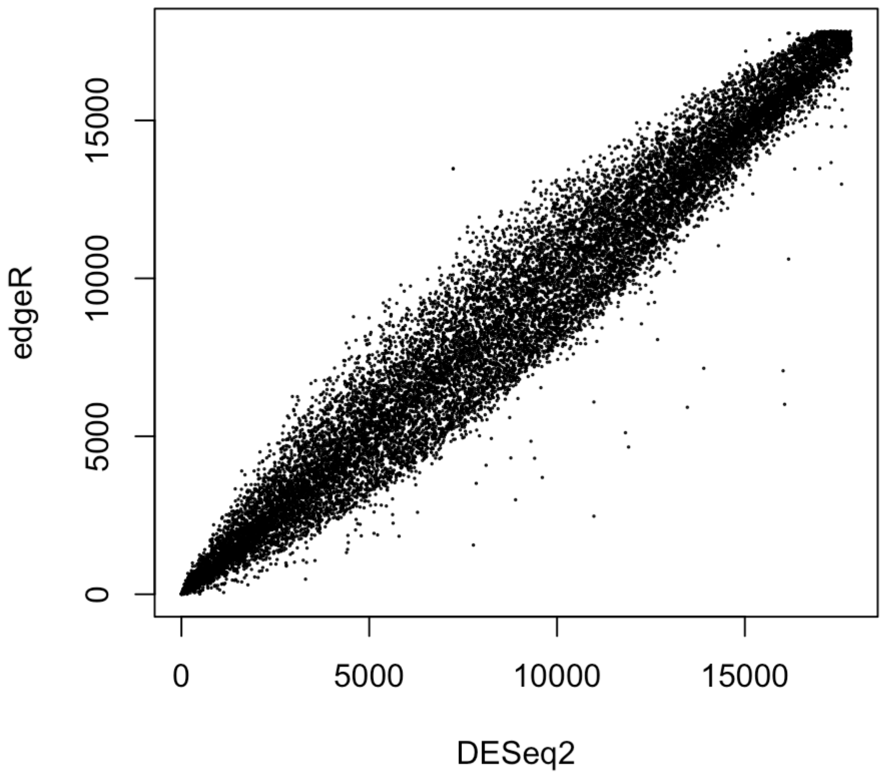
Also with edgeR we can test for significance relative to a fold-change threshold, using the function `glmTreat` instead of `glmLRT`. Below we set the log fold-change threshold to 1 (i.e., fold change threshold equal to 2), as for DESeq2 above. We also compare the results to those obtained with DESeq2.

```
treatres <- glmTreat(fit, coef = ncol(design), lfc = 1)
tt.treat <- topTags(treatres, n = nrow(y), sort.by = "none")
table(DESeq2 = resLFC1$padj < 0.1, edgeR = tt.treat$table$FDR < 0.1)
```

```
##        edgeR
## DESeq2  FALSE  TRUE
##   FALSE 18178   424
##   TRUE      0   196
```

We can compare the two lists by the ranks:

```
common <- !is.na(res$padj)
plot(rank(res$padj[common]),
     rank(tt.all$table$FDR[common]), cex=.1,
     xlab="DESeq2", ylab="edgeR")
```



Multiple testing

In high-throughput biology, we are careful to not use the *p* values directly as evidence against the null, but to correct for *multiple testing*. What would happen if we were to simply threshold the *p* values at a low value, say 0.05? There are 5666 genes with a *p* value below 0.05 among the 25133 genes, for which the test succeeded in reporting a *p* value:

```
sum(res$pvalue < 0.05, na.rm=TRUE)
```

```
## [1] 5666
```

```
sum(!is.na(res$pvalue))
```

```
## [1] 25133
```

Now, assume for a moment that the null hypothesis is true for all genes, i.e., no gene is affected by the treatment with dexamethasone. Suppose we are interesting in a significance level of 0.05. Then, by the definition of the *p* value, we expect up to 5% of the genes to have a *p* value below 0.05. This amounts to

```
round(sum(!is.na(res$pvalue)) * .05)
```

File failed to load: /extensions/MathZoom.js

```
## [1] 1257
```

If we just considered the list of genes with a *p* value below 0.05 as differentially expressed, this list should therefore be expected to contain many false positives:

```
round(sum(!is.na(res$pvalue)) * .05) # expected 'null' less than 0.05
```

```
## [1] 1257
```

```
sum(res$pvalue < .05, na.rm=TRUE) # observed p < .05
```

```
## [1] 5666
```

```
# expected ratio of false positives in the set with p < .05
round(sum(!is.na(res$pvalue))*.05 / sum(res$pvalue < .05, na.rm=TRUE), 2)
```

```
## [1] 0.22
```

DESeq2 and *edgeR* use the Benjamini-Hochberg (BH) adjustment (Benjamini and Hochberg 1995) as implemented in the base R *p.adjust* function; in brief, this method calculates for each gene an adjusted *p* value that answers the following question: if one called significant all genes with an adjusted *p* value less than or equal to this gene's adjusted *p* value threshold, what would be the fraction of false positives (the *false discovery rate*, FDR) among them, in the sense of the calculation outlined above? These values, called the BH-adjusted *p* values, are given in the column `padj` of the `res` object.

The FDR is a useful statistic for many high-throughput experiments, as we are often interested in reporting or focusing on a set of interesting genes, and we would like to put an upper bound on the percent of false positives in this set.

Hence, if we consider a fraction of 10% false positives acceptable, we can consider all genes with an adjusted *p* value below 10% = 0.1 as significant. How many such genes are there?

```
sum(res$padj < 0.1, na.rm=TRUE)
```

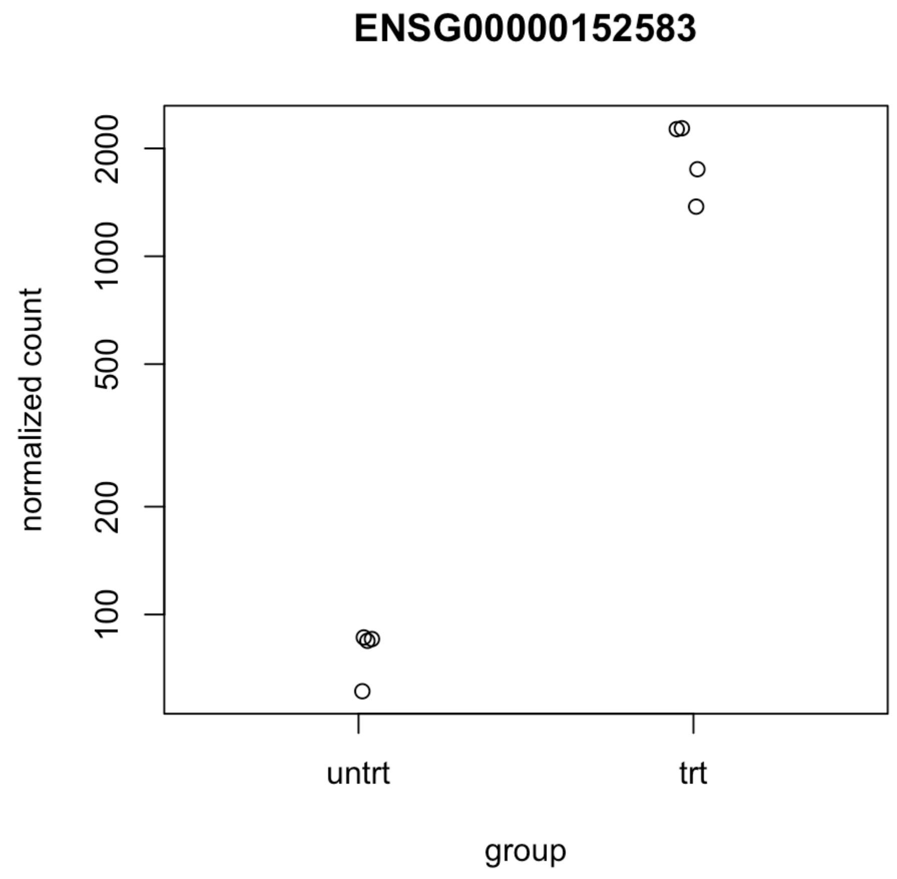
```
## [1] 4846
```

Plotting results

A quick way to visualize the counts for a particular gene is to use the *plotCounts* function that takes as arguments the *DESeqDataSet*, a gene name, and the group over which to plot the counts (Figure below).

```
topGene <- rownames(res)[which.min(res$padj)]
plotCounts(dds, topGene, "dex")
```

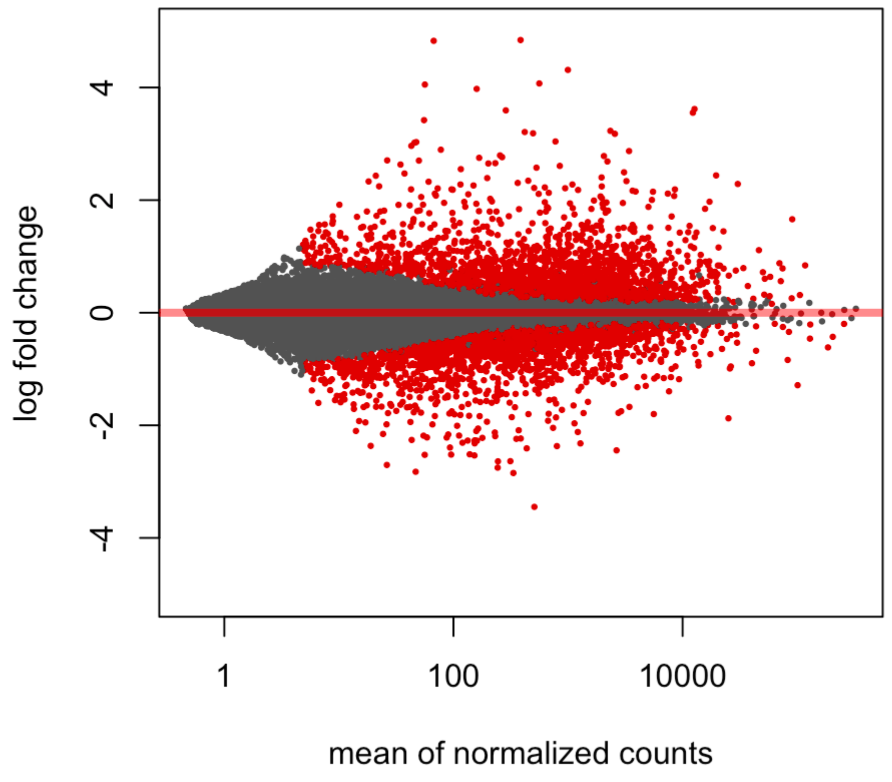
File failed to load: /extensions/MathZoom.js



MA plot with DESeq2

An *MA-plot* (Dudoit et al. 2002) provides a useful overview for an experiment with a two-group comparison (Figure below).

```
DESeq2::plotMA(res, ylim=c(-5,5))
```



The log2 fold change for a particular comparison is plotted on the y-axis and the average of the counts normalized by size factor is shown on the x-axis ("M" for minus, because a log ratio is equal to log minus log, and "A" for average). Each gene is represented with a dot. Genes with an adjusted *p* value below a threshold (here 0.1, the default) are shown in red.

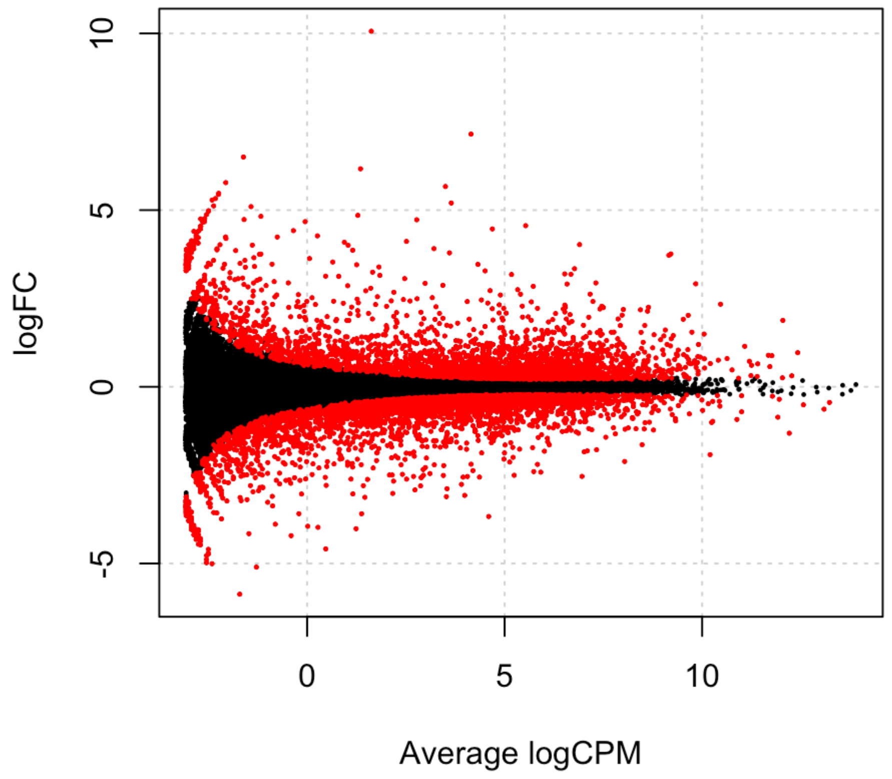
The *DESeq2* package uses statistical techniques to moderate log2 fold changes from genes with very low counts and highly variable counts, as can be seen by the narrowing of the vertical spread of points on the left side of the MA-plot. For a detailed explanation of the rationale of moderated fold changes, please see the *DESeq2* paper (Love, Huber, and Anders 2014). This plot demonstrates that only genes with a large average normalized count contain sufficient information to yield a significant call.

MA / Smear plot with edgeR

In *edgeR*, the MA plot is obtained via the `plotSmear` function.

```
plotSmear(lrt, de.tags=tt$table$gene.id)
```

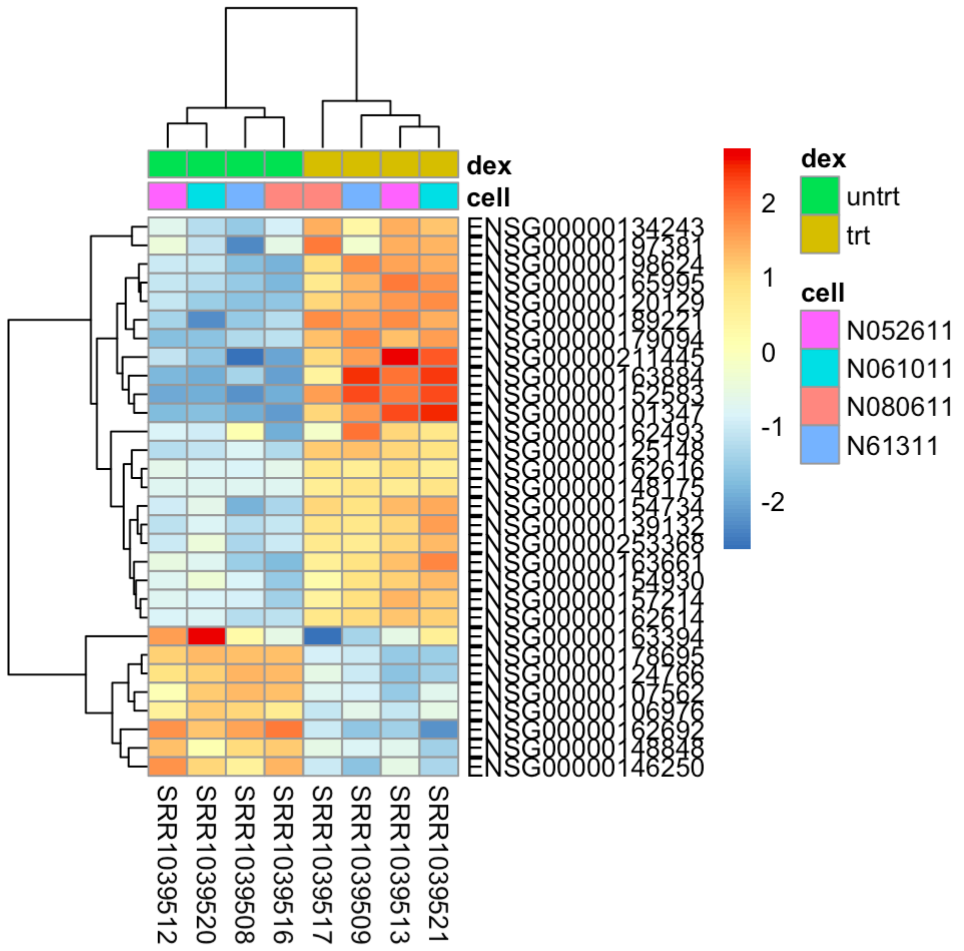
File failed to load: /extensions/MathZoom.js



Heatmap of the most significant genes

```
library("pheatmap")
mat <- assay(vsd)[ head(order(res$padj),30), ]
mat <- mat - rowMeans(mat)
df <- as.data.frame(colData(vsd)[,c("cell","dex")])
pheatmap(mat, annotation_col=df)
```

File failed to load: /extensions/MathZoom.js



Annotating and exporting results

Our result table so far only contains Ensembl gene IDs, but actual gene names may be more informative for interpretation. Bioconductor's annotation packages help with mapping various ID schemes to each other. We load the *AnnotationDbi (http://bioconductor.org/packages/AnnotationDbi)* package and the annotation package *Homo.sapiens (http://bioconductor.org/packages/Homo.sapiens)*:

```
library("AnnotationDbi")
library("Homo.sapiens")
```

To get a list of all available key types, use:

```
columns(Homo.sapiens)
```

File failed to load: /extensions/MathZoom.js

```
##  [1] "ACCNUM"       "ALIAS"        "CDSCHROM"     "CDSEND"       "CDSID"        "CDSN
AME"
##  [7] "CDSSTART"     "CDSSTRAND"    "DEFINITION"   "ENSEMBL"      "ENSEMBLPROT"  "ENSE
MBLTRANS"
## [13] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "EXONCHROM"    "EXON
END"
## [19] "EXONID"       "EXONNAME"     "EXONRANK"     "EXONSTART"    "EXONSTRAND"   "GENE
ID"
## [25] "GENENAME"     "GO"           "GOALL"        "GOID"         "IPI"          "MAP"

## [31] "OMIM"         "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"         "PMI
D"
## [37] "PROSITE"      "REFSEQ"       "SYMBOL"       "TERM"         "TXCHROM"      "TXEN
D"
## [43] "TXID"         "TXNAME"       "TXSTART"      "TXSTRAND"     "TXTYPE"       "UCSC
KG"
## [49] "UNIGENE"      "UNIPROT"
```

We can use the *mapIds* function to add individual columns to our results table. We provide the row names of our results table as a key, and specify that `keytype=ENSEMBL`. The `column` argument tells the *mapIds* function which information we want, and the `multiVals` argument tells the function what to do if there are multiple possible values for a single input value. Here we ask to just give us back the first one that occurs in the database. To add the gene symbol and Entrez ID, we call *mapIds* twice.

```
res$symbol <- mapIds(Homo.sapiens,
                     keys=row.names(res),
                     column="SYMBOL",
                     keytype="ENSEMBL",
                     multiVals="first")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
y$genes$symbol <- res$symbol

res$entrez <- mapIds(Homo.sapiens,
                     keys=row.names(res),
                     column="ENTREZID",
                     keytype="ENSEMBL",
                     multiVals="first")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
y$genes$entrez <- res$entrez

res$symbol <- mapIds(Homo.sapiens,
                     keys=row.names(res),
                     column="GENENAME",
                     keytype="ENSEMBL",
                     multiVals="first")
```

File failed to load: /extensions/MathZoom.js

```
## 'select()' returned 1:many mapping between keys and columns
```

```
y$genes$symbol <- res$symbol
```

Now the results have the desired external gene IDs:

```
resOrdered <- res[order(res$padj),]
head(resOrdered)
```

```
## log2 fold change (MAP): dex trt vs untrt
## Wald test p-value: dex trt vs untrt
## DataFrame with 6 rows and 8 columns
##                   baseMean log2FoldChange     lfcSE      stat        pvalue
 padj
##                  <numeric>      <numeric> <numeric> <numeric>     <numeric>     <nume
ric>
## ENSG00000152583   997.4398       4.312475 0.1724161  25.01203 4.523551e-138 8.062777e
-134
## ENSG00000165995   495.0929       3.186993 0.1279073  24.91643 4.938669e-137 4.401342e
-133
## ENSG00000120129  3409.0294       2.871149 0.1183764  24.25441 5.940410e-130 3.529396e
-126
## ENSG00000101347 12703.3871       3.617657 0.1493023  24.23041 1.063867e-129 4.740593e
-126
## ENSG00000189221  2341.7673       3.229713 0.1368981  23.59210 4.645010e-123 1.655853e
-119
## ENSG00000211445 12285.6151       3.551995 0.1583908  22.42551 2.219353e-111 6.592957e
-108
##                                                                symbol      entrez
##                                                           <character> <character>
## ENSG00000152583                                               SPARC like 1        8404
## ENSG00000165995 calcium voltage-gated channel auxiliary subunit beta 2         783
## ENSG00000120129                             dual specificity phosphatase 1        1843
## ENSG00000101347                                 SAM domain and HD domain 1       25939
## ENSG00000189221                                        monoamine oxidase A        4128
## ENSG00000211445                                  glutathione peroxidase 3        2878
```

Exporting results to CSV file

You can easily save the results table in a CSV file, that you can then share or load with a spreadsheet program such as Excel. The call to *as.data.frame* is necessary to convert the *DataFrame* object (*IRanges (http://bioconductor.org/packages/IRanges)* package) to a *data.frame* object that can be processed by *write.csv*. Here, we take just the top 100 genes for demonstration.

```
resOrderedDF <- as.data.frame(resOrdered)[seq_len(100),]
write.csv(resOrderedDF, file="results.csv")
```

Exporting results to Glimma

Glimma (http://bioconductor.org/packages/Glimma) is a new package in Bioconductor which allows one to build interactive HTML pages that summarize the results of an edgeR, limma or DESeq2 analysis. See the *Glimma* vignette for more details on how to customize this HTML page and what other plots are available.

Note that the Glimma functions take a minute to build.

edgeR results:

```
library("Glimma")
glMDPlot(lrt,
         counts=y$counts,
         anno=y$genes,
         groups=y$samples$dex,
         samples=colnames(y),
         status=tt.all$table$FDR < 0.1,
         id.column="gene.id")
```

DESeq2 results:

```
res.df <- as.data.frame(res)
res.df$log10MeanNormCount <- log10(res.df$baseMean)
idx <- rowSums(counts(dds)) > 0
res.df <- res.df[idx,]
res.df$padj[is.na(res.df$padj)] <- 1
glMDPlot(res.df,
         xval="log10MeanNormCount",
         yval="log2FoldChange",
         counts=counts(dds)[idx,],
         anno=data.frame(GeneID=rownames(dds)[idx]),
         groups=dds$dex,
         samples=colnames(dds),
         status=res.df$padj < 0.1,
         display.columns=c("symbol", "entrez"))
```

Gene set overlap analysis

Now that we have obtained a list of genes that are significantly up- or down-regulated in response to treatment, a natural question is: What do these genes have in common? A simple way to answer this is to compare with a gene set collection, i.e., a collection of sets of genes, which have something specific in common. For each such set, we ask? Are the genes in this set significantly enriched in our list of down-regulated genes, i.e., do they appear more often in this list as one would expect by chance? As explained in the lecture, the hypergeometric test (also called Fisher's test) is often used for this purpose.

The Gene Ontology (GO) is a controlled vocabular of terms used to describe what it known about genes' function with respect to three aspects, namely their molecular function (MF), the biological process (BP) they play a role in, and the cellular component (CC) the gene's product is part of. The annotation package *org.Hs.eg.db (http://bioconductor.org/packages/org.Hs.eg.db)* (part of the package *Homo.sapiens (http://bioconductor.org/packages/Homo.sapiens)*, which we have already used above), contains mappings of human genes to GO terms. The *topGO (http://bioconductor.org/packages/topGO)* package (Alexa, Rahnenfuhrer, and Lengauer 2006) offers a simple way to perform enrichment tests for all the GO terms, which we demonstrate now.

First, we should subset our result table to only those genes that had suffient read coverage that we could actually test for differential expression. In DESeq2, we can simply take the genes that survived independent filtering (see above) and hence got assigned an adjusted p value.

We will use the set of genes that had evidence of a fold change greater than 2 (log2 fold change greater than 1):

```
resTested <- resLFC1[ !is.na(resLFC1$padj), ]
```

Then, we construct a factor containing ones and zeroes to indicate which of these genes were significantly up-regulated and which not. We name the vector elements with the Ensembl gene IDs.

```
genelistUp <- factor( as.integer( resTested$padj < .1 & resTested$log2FoldChange > 0 ) )
names(genelistUp) <- rownames(resTested)
```

Now, we load the topGO package and prepare a data structure for testing.

```
library("topGO")

myGOdata <- new( "topGOdata",
   ontology = "BP",
   allGenes = genelistUp,
   nodeSize = 10,
   annot = annFUN.org, mapping = "org.Hs.eg.db", ID="ensembl" )
```

Here, we have indicated that we want to work within the Biological Processes (BP) sub-ontology, use only sets with at least 10 genes, and that the names of our gene list are Ensembl IDs, which are mapped to GO terms in the `org.Hs.eg.db` annotation database object.

We run Fisher tests and choose topGO's "elim" algorithm, which eliminates broader terms (like, e.g., "mitochondrion") if a more narrow term (e.g., "mitochandrial membrane") can be found to describe the enrichment:

```
goTestResults <- runTest( myGOdata, algorithm = "elim", statistic = "fisher" )
```

```
##
##            -- Elim Algorithm --
##
##        the algorithm is scoring 2627 nontrivial nodes
##        parameters:
##            test statistic: fisher
##            cutOff: 0.01
```

```
##
##   Level 17:  2 nodes to be scored    (0 eliminated genes)
```

```
##
##   Level 16:  5 nodes to be scored    (0 eliminated genes)
```

File failed to load: /extensions/MathZoom.js

```
##
##   Level 15:  13 nodes to be scored   (0 eliminated genes)
```

```
##
##   Level 14:  30 nodes to be scored   (0 eliminated genes)
```

```
##
##   Level 13:  56 nodes to be scored   (0 eliminated genes)
```

```
##
##   Level 12:  95 nodes to be scored   (43 eliminated genes)
```

```
##
##   Level 11:  154 nodes to be scored  (98 eliminated genes)
```

```
##
##   Level 10:  220 nodes to be scored  (235 eliminated genes)
```

```
##
##   Level 9:   314 nodes to be scored  (255 eliminated genes)
```

```
##
##   Level 8:   366 nodes to be scored  (400 eliminated genes)
```

```
##
##   Level 7:   436 nodes to be scored  (751 eliminated genes)
```

```
##
##   Level 6:   397 nodes to be scored  (1528 eliminated genes)
```

```
##
##   Level 5:   295 nodes to be scored  (2060 eliminated genes)
```

```
##
##   Level 4:   167 nodes to be scored  (2167 eliminated genes)
```

```
##
##   Level 3:   56 nodes to be scored   (2295 eliminated genes)
```

```
##
##   Level 2:   20 nodes to be scored   (4181 eliminated genes)
```

File failed to load: /extensions/MathZoom.js

```
##
##   Level 1:   1 nodes to be scored    (4181 eliminated genes)
```

```
GenTable( myGOdata, goTestResults )
```

```
##         GO.ID                                       Term Annotated Significant Expec
ted result1
## 1  GO:0071294               cellular response to zinc ion        11           4
 0.10 2.3e-06
## 2  GO:0035358 regulation of peroxisome proliferator ac...        10           3
 0.09 9.2e-05
## 3  GO:0044320        cellular response to leptin stimulus        13           3
 0.12 0.00022
## 4  GO:0045926               negative regulation of growth       191           8
 1.79 0.00043
## 5  GO:1903792      negative regulation of anion transport        19           3
 0.18 0.00070
## 6  GO:0045598      regulation of fat cell differentiation        86           5
 0.81 0.00127
## 7  GO:0044060             regulation of endocrine process        25           3
 0.23 0.00160
## 8  GO:0072358           cardiovascular system development       770          16
 7.23 0.00219
## 9  GO:0060986                 endocrine hormone secretion        28           3
 0.26 0.00223
## 10 GO:0003044 regulation of systemic arterial blood pr...        28           3
 0.26 0.00223
```

Inspect the table. Which results make sense, which don't? Try again for the other two sub-ontologies, and for the down-regulated genes. Remember that gene set enrichments always have to be taken with a grain of salt, but can be a good starting point for further downstream analysis of a gene list.

Session information

```
sessionInfo()
```

File failed to load: /extensions/MathZoom.js

```
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices datasets  utils     methods   b
ase
##
## other attached packages:
##  [1] Homo.sapiens_1.3.1                      TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] org.Hs.eg.db_3.3.0                      OrganismDbi_1.14.1
##  [5] pheatmap_1.0.8                          PoiClaClu_1.0.2
##  [7] ggplot2_2.1.0                           edgeR_3.14.0
##  [9] limma_3.28.10                           Rsubread_1.22.2
## [11] BiocParallel_1.6.2                      GenomicAlignments_1.8.3
## [13] GenomicFeatures_1.24.2                  Rsamtools_1.24.0
## [15] Biostrings_2.40.2                       XVector_0.12.0
## [17] airway_0.106.2                          BiocStyle_2.0.2
## [19] topGO_2.24.0                            SparseM_1.7
## [21] GO.db_3.3.0                             AnnotationDbi_1.34.3
## [23] graph_1.50.0                            rmarkdown_0.9.6.14
## [25] DESeq2_1.12.3                           SummarizedExperiment_1.2.3
## [27] Biobase_2.32.0                          GenomicRanges_1.24.2
## [29] GenomeInfoDb_1.8.1                      IRanges_2.6.1
## [31] S4Vectors_0.10.1                        BiocGenerics_0.18.0
## [33] magrittr_1.5                            knitr_1.13
## [35] testthat_1.0.2                          devtools_1.11.1
## [37] BiocInstaller_1.22.3
##
## loaded via a namespace (and not attached):
##  [1] splines_3.3.0       Formula_1.2-1       latticeExtra_0.6-28 RBGL_1.48.1
##  [5] yaml_2.1.13         RSQLite_1.0.0       lattice_0.20-33     chron_2.3-47
##  [9] digest_0.6.9        RColorBrewer_1.1-2  colorspace_1.2-6    htmltools_0.3.5
## [13] Matrix_1.2-6        plyr_1.8.4          XML_3.98-1.4        biomaRt_2.28.0
## [17] genefilter_1.54.2   zlibbioc_1.18.0     xtable_1.8-2        scales_0.4.0
## [21] annotate_1.50.0     withr_1.0.2         nnet_7.3-12         survival_2.39-4
## [25] crayon_1.3.1        memoise_1.0.0       evaluate_0.9        foreign_0.8-66
## [29] tools_3.3.0         data.table_1.9.6    formatR_1.4         matrixStats_0.50.2
## [33] stringr_1.0.0       munsell_0.4.3       locfit_1.5-9.1      cluster_2.0.4
## [37] grid_3.3.0          RCurl_1.95-4.8      labeling_0.3        bitops_1.0-6
## [41] gtable_0.2.0        DBI_0.4-1           R6_2.1.2            gridExtra_2.2.1
## [45] rtracklayer_1.32.1  Hmisc_3.17-4        stringi_1.1.1       Rcpp_0.12.5
## [49] geneplotter_1.50.0  rpart_4.1-10        acepack_1.3-3.3
```

References

Alexa, A., J. Rahnenfuhrer, and T. Lengauer. 2006. “Improved Scoring of Functional Groups from Gene Expression Data by Decorrelating GO Graph Structure.” *Bioinformatics* 22 (13): 1600–1607. doi:10.1093/bioinformatics/btl140 (https://doi.org/10.1093/bioinformatics/btl140).

File failed to load: /extensions/MathZoom.js

Anders, Simon, Paul T. Pyl, and Wolfgang Huber. 2015. “HTSeq – a Python framework to work with high-throughput sequencing data.” *Bioinformatics* 31 (2). Oxford University Press: 166–69. doi:10.1093/bioinformatics/btu638 (https://doi.org/10.1093/bioinformatics/btu638).

Benjamini, Yoav, and Yosef Hochberg. 1995. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” *Journal of the Royal Statistical Society. Series B (Methodological)* 57 (1): 289–300. http://www.jstor.org/stable/2346101 (http://www.jstor.org/stable/2346101).

Bray, Nicolas, Harold Pimentel, Pall Melsted, and Lior Pachter. 2015. “Near-Optimal RNA-Seq Quantification.” *ArXiv*. http://arxiv.org/abs/1505.02710 (http://arxiv.org/abs/1505.02710).

Dobin, Alexander, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. 2013. “STAR: ultrafast universal RNA-seq aligner.” *Bioinformatics* 29 (1). Oxford University Press: 15–21. doi:10.1093/bioinformatics/bts635 (https://doi.org/10.1093/bioinformatics/bts635).

Dudoit, Rine, Yee H. Yang, Matthew J. Callow, and Terence P. Speed. 2002. “Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments.” In *Statistica Sinica*, 111–39. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.9702 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.9702).

Durinck, Steffen, Paul T. Spellman, Ewan Birney, and Wolfgang Huber. 2009. “Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt.” *Nature Protocols* 4 (8). Nature Publishing Group: 1184–91. doi:10.1038/nprot.2009.97 (https://doi.org/10.1038/nprot.2009.97).

Flicek, Paul, M. Ridwan Amode, Daniel Barrell, Kathryn Beal, Konstantinos Billis, Simon Brent, Denise Carvalho-Silva, et al. 2014. “Ensembl 2014.” *Nucleic Acids Research* 42 (D1). Oxford University Press: D749–D755. doi:10.1093/nar/gkt1196 (https://doi.org/10.1093/nar/gkt1196).

Hardcastle, Thomas, and Krystyna Kelly. 2010. “baySeq: Empirical Bayesian methods for identifying differential expression in sequence count data.” *BMC Bioinformatics* 11 (1): 422+. doi:10.1186/1471-2105-11-422 (https://doi.org/10.1186/1471-2105-11-422).

Himes, Blanca E., Xiaofeng Jiang, Peter Wagner, Ruoxi Hu, Qiyu Wang, Barbara Klanderman, Reid M. Whitaker, et al. 2014. “RNA-Seq transcriptome profiling identifies CRISPLD2 as a glucocorticoid responsive gene that modulates cytokine function in airway smooth muscle cells.” *PloS One* 9 (6). doi:10.1371/journal.pone.0099625 (https://doi.org/10.1371/journal.pone.0099625).

Huber, Wolfgang, Vincent J. Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S. Carvalho, Hector Corrada C. Bravo, et al. 2015. “Orchestrating high-throughput genomic analysis with Bioconductor.” *Nature Methods* 12 (2). Nature Publishing Group: 115–21. doi:10.1038/nmeth.3252 (https://doi.org/10.1038/nmeth.3252).

Kent, W. James, Charles W. Sugnet, Terrence S. Furey, Krishna M. Roskin, Tom H. Pringle, Alan M. Zahler, and David Haussler. 2002. “The human genome browser at UCSC.” *Genome Research* 12 (6). Cold Spring Harbor Laboratory Press: 996–1006. doi:10.1101/gr.229102 (https://doi.org/10.1101/gr.229102).

Law, Charity W., Yunshun Chen, Wei Shi, and Gordon K. Smyth. 2014. “Voom: precision weights unlock linear model analysis tools for RNA-seq read counts.” *Genome Biology* 15 (2). BioMed Central Ltd: R29+. doi:10.1186/gb-2014-15-2-r29 (https://doi.org/10.1186/gb-2014-15-2-r29).

Lawrence, Michael, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, and Vincent J. Carey. 2013. “Software for Computing and Annotating Genomic Ranges.” Edited by Andreas Prlic. *PLoS Computational Biology* 9 (8). Public Library of Science: e1003118+. doi:10.1371/journal.pcbi.1003118 (https://doi.org/10.1371/journal.pcbi.1003118).

Leng, N., J. A. Dawson, J. A. Thomson, V. Ruotti, A. I. Rissman, B. M. G. Smits, J. D. Haag, M. N. Gould, R. M. Stewart, and C. Kendziorski. 2013. “EBSeq: an empirical Bayes hierarchical model for inference in RNA-seq experiments.” *Bioinformatics* 29 (8). Oxford University Press: 1035–43. doi:10.1093/bioinformatics/btt087 (https://doi.org/10.1093/bioinformatics/btt087).

Li, Bo, and Colin N. Dewey. 2011. “RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome.” *BMC Bioinformatics* 12: 323+. doi:10.1186/1471-2105-12-3231 (https://doi.org/10.1186/1471-2105-12-3231).

Liao, Y., G. K. Smyth, and W. Shi. 2014. “featureCounts: an efficient general purpose program for assigning sequence reads to genomic features.” *Bioinformatics* 30 (7). Oxford University Press: 923–30. doi:10.1093/bioinformatics/btt656 (https://doi.org/10.1093/bioinformatics/btt656).

Love, Michael I., Simon Anders, Vladislav Kim, and Wolfgang Huber. 2015. “RNA-Seq Workflow: Gene-Level Exploratory Analysis and Differential Expression.” *F1000Research*, October. doi:10.12688/f1000research.7035.1 (https://doi.org/10.12688/f1000research.7035.1).

Love, Michael I., Wolfgang Huber, and Simon Anders. 2014. “Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.” *Genome Biology* 15 (12). BioMed Central Ltd: 550+. doi:10.1186/s13059-014-0550-8 (https://doi.org/10.1186/s13059-014-0550-8).

Patro, Rob, Geet Duggal, and Carl Kingsford. 2015. “Salmon: Accurate, Versatile and Ultrafast Quantification from RNA-Seq Data Using Lightweight-Alignment.” *BioRxiv*. http://biorxiv.org/content/early/2015/06/27/021592 (http://biorxiv.org/content/early/2015/06/27/021592).

Patro, Rob, Stephen M. Mount, and Carl Kingsford. 2014. “Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms.” *Nature Biotechnology* 32: 462–64. doi:10.1038/nbt.2862 (https://doi.org/10.1038/nbt.2862).

Robert, Christelle, and Mick Watson. 2015. “Errors in RNA-Seq quantification affect genes of relevance to human disease.” *Genome Biology*. doi:10.1186/s13059-015-0734-x (https://doi.org/10.1186/s13059-015-0734-x).

Robinson, M. D., D. J. McCarthy, and G. K. Smyth. 2009. “edgeR: a Bioconductor package for differential expression analysis of digital gene expression data.” *Bioinformatics* 26 (1). Oxford University Press: 139–40. doi:10.1093/bioinformatics/btp616 (https://doi.org/10.1093/bioinformatics/btp616).

Soneson, Charlotte, Michael I. Love, and Mark Robinson. 2015. “Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences.” *F1000Research* 4 (1521). doi:10.12688/f1000research.7563.1 (https://doi.org/10.12688/f1000research.7563.1).

Trapnell, Cole, David G Hendrickson, Martin Sauvageau, Loyal Goff, John L Rinn, and Lior Pachter. 2013. “Differential analysis of gene regulation at transcript resolution with RNA-seq.” *Nature Biotechnology*. doi:10.1038/nbt.2450 (https://doi.org/10.1038/nbt.2450).

Wickham, Hadley. 2009. *ggplot2*. New York, NY: Springer New York. doi:10.1007/978-0-387-98141-3 (https://doi.org/10.1007/978-0-387-98141-3).

Wu, Hao, Chi Wang, and Zhijin Wu. 2013. “A new shrinkage estimator for dispersion improves differential expression detection in RNA-seq data.” *Biostatistics* 14 (2). Oxford University Press: 232–43. doi:10.1093/biostatistics/kxs033 (https://doi.org/10.1093/biostatistics/kxs033).