deltaRpkm

An R pacakge for rapid detection of differential presence of genes associated with a phenotype

Hatice Akarsu[1,2], Lisandra Aguilar-Bultet[3], Laurent Falquet[1,2]

[1]Department of Biology, University of Fribourg, Switzerland,
[2]Swiss Institute of Bioinformatics, BUGFri group, Fribourg, Switzerland
[3]Institute of Veterinary Bacteriology, University of Bern, Switzerland,

Feb 2019

deltaRpkm *is an R package whose main purpose is to quickly identify genes potentially involved in a given phenotype by performing a differential analysis of genes coverage between two sets of closely related genomes. The package provides functions to compute the RPKM, the $\delta RPKM$, candidate genes filtering and heatmap plot. It also includes methods to perform some batch effects controls and diagnostics plots.*

Contents

1 Prerequisites

1.1 Input files

The deltaRpkm package requires 2 user input files:

1. a metadata table that provides parameters information for the groups comparisons, with the following mandatory fields:

   ```
   <sample> <phenotype_1> <phenotype_2>  <genome_length> <mapped_reads> <...>
   ```

   - <**sample**> the column containing the samples names
   - <**phenotype_1**> the column containing the column name of the phenotype being investigated for the genes differential presence analysis
   - <**phenotype_2**> the column containing the 2nd phenotype that can be added on the heatmap for comparison
   - <**genome_length**> the column containing the reference genome length
   - <**mapped_reads**> the column containing the total number of mapped reads in each sample
   - <**...**>

   These are the minimum required elements to be given to the pipeline; more factors can be included for alternative analysis if desired.

2. a coverage table that combines the mapped reads counts per gene and per sample, as this:

   ```
   <chr> <start> <end> <geneID> <sample1_readsCounts> <sample2_readsCounts> <...>
   ```

   - <**chr**> the column containing the chromosome name
   - <**start**> the column containing the gene start coordinate
   - <**end**> the column containing the gene end coordinate
   - <**geneID**> the column containing the gene identifier
   - <**sample1_readsCounts**> the column containing the mapped reads counts of sample1
   - <**sample2_readsCounts**> the column containing the mapped reads counts of sample2
   - <**...**>

Please make sure that the input tables follow as much as possible those formats (columns order and names for the minimum required information). For instance, the samples names in the <sample> column of metadata table MUST be the same as the <sample_readsCounts> ones in the coverage table.

The working examples provided by the package correspond to datasets of different sizes from *Listeria monocytogenes* (Aguilar-Bultet *et al.*, 2018).

1.2 Minimum of R knowledge

Although deltaRpkm is very simple/user-friendly and comes with an extensive documentation (`https://github.com/frihaka/deltaRpkm/tree/master/doc`), it assumes a minimum familiarity with R, *e.g* installing libraries from CRAN and Bioconductor, awareness of working environment, basic R commands and objects etc.

1.3 R dependencies

The R package deltaRpkm, like any R packages, is built upon common R **CRAN** libraries that should be already installed in your system if you are an R user (ggplot2, ggfortify, dplyr...).

It also requires some **Bioconductor** R packages. If not already in your system, please make sure that the following R packages from Bioconductor - sva and Biostrings - are installed:

```
# for R >= 3.5
> if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("sva", version = "3.8")
BiocManager::install("Biostrings", version = "3.8")

# for R <= 3.4
source("https://bioconductor.org/biocLite.R")
biocLite("sva")
biocLite("Biostrings")
```

1.4 Download and install deltaRpkm

From GitHub https://github.com/frihaka/deltaRpkm/, download the compressed binary file on a local directory.

If you are familiar with R and have already most of the common R packages (ggplot2, ggfortify, dplyr...), you can install from the terminal with the `R CMD INSTALL` command:

```
R CMD INSTALL deltaRpkm_0.1.0_R_x86_64-pc-linux-gnu.tar.gz
```

and then just install the Bioconductor packages as shown above.
If you are not very familiar with R ecosystem, try rather to install deltaRpkm with the funciton `install.packages()`, from inside R/RStudio:

```
> install.packages("path/2/deltaRpkm_0.1.0_R_x86_64-pc-linux-gnu.tar.gz",
    repos = NULL,
    dependencies = TRUE)
```

This will download your missing CRAN libraries that are required by deltaRpkm. You will still need to install the Bioconductor packages, as shown above.

2 Overview of the pipeline

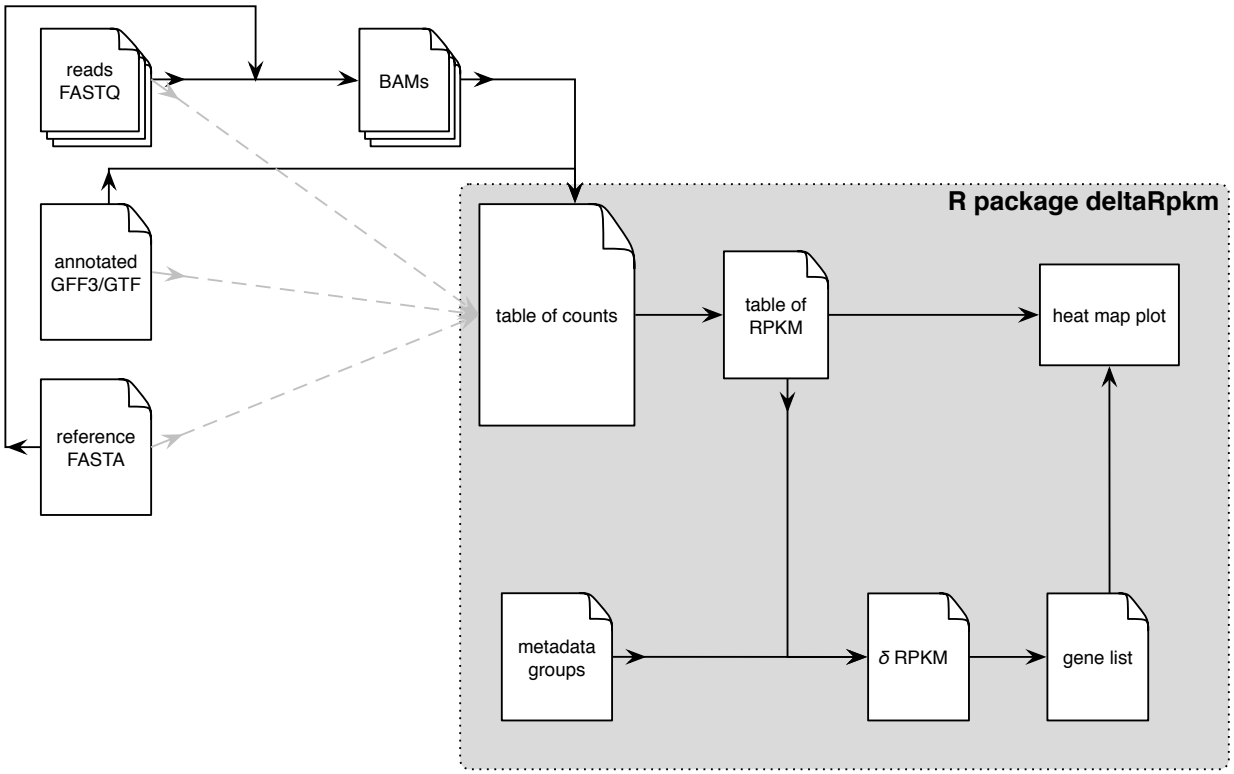


Figure 1: Overview of the deltaRpkm pipeline.

3 Building and loading the coverage table

The reads counts per gene table must be pre-computed and provided by the user. Below is the command used to build it with `bedtools multicov` on a terminal:

```
bedtools multicov -bams aln.1.bam ... aln.n.bam -bed <bed/gff/vcf> > coverage_table.csv
```

Notes on `bedtools multicov`:

- any number of bam files can be run together in a batch mode, thus allowing all the samples of the dataset to be included in the coverage table
- do not forget to redirect/output the results into a coverage table

The user can build its reads counts table with other methods, like the RNA-seq aligner called STAR that mapps and produces the coverage table at once.

Please do not forget to ensure that your custom coverage table (either produced by bedtools multicov or by STAR etc) follows the following required format:

- **<chr>**
- **<start>**
- **<end>**
- **<geneID>**
- **<sample1_readsCounts>**
- **<sample2_readsCounts>**
- <...>

Alternatively, example datasets derived from Aguilar-Bultet *et al.*, 2018 are available in the deltaRpkm package:

```
> data("coverage_table_N51") # this creates coverage_table df in the
    environment
> head(coverage_table_N51[, 1:8])
                chr start  end         geneID JF4931 JF5172 JF5761
                   JF5827
1 JF5203_chromosome   318 1674 LMJF5203_00001   3109   1466   5582
   5761
2 JF5203_chromosome  1867 3013 LMJF5203_00002   2778   1099   4737
   4882
3 JF5203_chromosome  3120 4464 LMJF5203_00003   3218   1473   4914
   5365
4 JF5203_chromosome  4577 4865 LMJF5203_00004    947    358   1568
   1546
5 JF5203_chromosome  4868 5981 LMJF5203_00005   2578    932   4415
   4572
6 JF5203_chromosome  6029 7970 LMJF5203_00006   4125   1853   7018
   7681
```

4 Loading the metadata table

The user must provide a metadata table with some minimum informations/columns about the samples:

- a column containing the phenotype 1 data, that will be used for the RPKM comparisons. This is the main phenotype of interest being studied, *i.e* the phenotype of group 1 with the reference genome. This phenotype is the criteria of categorizing the datasets into 2 distinct groups and the basis of the whole comparison. For example: "neurovirulence"
- a column containing the phenotype 2 data, that will be used as a colored sidebar of the heatmap: this corresponds to a 2nd phenotype that the user can add to check whether the clustering in the heatmap correlates with this 2nd phenotype. For example: "infection_origin"

- reference genome length
- total number of mapped reads

The metadata/design table must be a data frame that looks like this:

Table 1: Example of input metadata table.

sample	platform	lineage_type	infection	genome_length	mapped_reads
JF4839	HiSeq2000	Lineage_II	ENV	2900890	8288011
JF4899	HiSeq4000	Lineage_I	CNS	2900890	9797440
JF4901	HiSeq2000	Lineage_I	CNS	2900890	1926369
JF4902	HiSeq2000	Lineage_I	CNS	2900890	1750981
JF4904	HiSeq2000	Lineage_I	CNS	2900890	1469430

A working example metadata dataset from deltaRpkm package is shown below:

```
> data("metadata_table_N51") # this creates metadata_table df in the
    env
> head(metadata_table_N51)
  sample  platform lineage_type infection genome_length mapped_reads
1 JF4906 HiSeq2000    Lineage_I       CNS       2900890      2042865
2 JF4929 HiSeq4000    Lineage_I       CNS       2900890      9469100
3 JF4931 HiSeq3000   Lineage_II       CNS       2900890      5285534
```

Format the metadata informations with `deltaRpkm::loadMetadata` function, giving in as arguments:

- **user_metadata** = <data frame of user input design table>
- **delta_phenotype_colname** = <phenotype 1 column name used to build the 2 categories>
- **heatmapbar_phenotype_colname** = <phenotype 2 column name used to build the extra bar in the heatmap>
- **samples_colname** = <column name containing the samples ID>
- **genome_length_colname** = <genomic length (in bp) of the reference genome used for mapping>
- **mapped_reads_colname** = <total number of mapped reads, for each sample ID>

```
> design_table <- loadMetadata(user_metadata = metadata_table_N51,
                               delta_phenotype_colname = "lineage_type
                                  ",
                               heatmapbar_phenotype_colname = "
                                  infection",
                               samples_colname = "sample",
```

```
                                genome_length_colname = "genome_length"
                                ,
                                mapped_reads_colname = "mapped_reads")

> head(design_table)
  sample lineage_type infection genome_length mapped_reads
1 JF4906    Lineage_I       CNS       2900890      2042865
2 JF4929    Lineage_I       CNS       2900890      9469100
3 JF4931   Lineage_II       CNS       2900890      5285534
```

5 Convert reads counts to RPKM values

5.1 RPKM formula

deltaRpkm uses the **Reads Per Kilobase Million RPKM** - a standard RNA-seq metrics that normalizes the reads counts per gene for **Sequencing Depth** and **Gene Length**:

with N_s being the total number of reads counts in the sample,

$$scalingFactor = \frac{N_s}{10^6} \tag{1}$$

$$RPM = \frac{ReadsCountsPerGene}{scalingFactor} \tag{2}$$

$$RPKM = \frac{RPM}{geneLength.10^{-3}} \tag{3}$$

The equation (2) corresponds to the normalization of the reads counts by the sample sequencing depth; and equation (3) to the normalization by the gene length.

5.2 Run `deltaRpkm::rpkm`

Run the following `deltaRpkm::rpkm` function to compute the RPKM values of each gene, in each sample:

```
> rpkmtable <- rpkm(user_metadata = design_table,
                    coverage_table = coverage_table_N51,
                    delta_phenotype_colname = "lineage_type",
                    heatmapbar_phenotype_colname = "infection")
> head(rpkmtable)
  sample         geneID lineage_type infection reads rpkm
1 JF4906 LMJF5203_00001    Lineage_I       CNS          1177  425
2 JF4906 LMJF5203_00002    Lineage_I       CNS           952  406
3 JF4906 LMJF5203_00003    Lineage_I       CNS          1080  393
```

6 $\delta RPKM$ values

6.1 $\delta RPKM$ formula

The analysis is centered around a pairwise comparison of genes presence/absence between genomes categorized into two different groups following the selected phenotype:

- a group 1 that shares the phenotype A of the reference genome
- a group 2 that does not have the reference phenotype A

For each pairwise comparison of a gene j between a genome x from group 1 and a genome y from group 2, `deltaRpkm::deltarpkm` function computes the difference of their RPKM values at gene j ($\delta RPKM_{j_{xy}}$) as:

$$\delta RPKM_{j_{xy}} = RPKM_{j_x} - RPKM_{j_y} \tag{4}$$

6.2 Run `deltaRpkm::deltarpkm`

```
> deltarpkm_table <- deltarpkm(rpkm_table = rpkmtable,
                               genes_names = unique(rpkmtable$geneID),
                               samples_colname = "sample",
                               delta_phenotype_colname = "lineage_type
                                  ",
                               reference_sample = "JF5203",
                               nonref_delta_phenotype = "Lineage_II")
> head(deltarpkm_table)
geneID sample.group1 lineage_type.group1 infection.group1 reads.group1
    rpkm.group1 sample.group2 lineage_type.group2 infection.group2
   reads.group2 rpkm.group2 deltarpkm
1 LMJF5203_00001         JF4906            Lineage_I              CNS
                    1177          425         JF4931          Lineage_II
                  CNS                     3109          433         -8
2 LMJF5203_00001         JF4906            Lineage_I              CNS
                    1177          425         JF5172          Lineage_II
                  CNS                     1466          520        -95
3 LMJF5203_00001         JF4906            Lineage_I              CNS
                    1177          425         JF5761          Lineage_II
                  ENV                     5582          465        -40
```

This run might take few minutes, depending on the size of datasets.

7 Differential genes presence

7.1 Strategy

The deltaRpkm package main feature is to screen for *the preferential presence of genes in the reference genome group, versus a comparison group.*
We use the method `deltaRpkm::deltaRPKMStats` to infer this set of genes, since they could potentially be involved in the reference genome phenotype (<lineage_type> ="Lineage_type_I"). This function:

1. computes for each gene j the **median value of all its** $\delta RPKM$ (m_j) derived from the samples pairwise comparisons. Note: a negative median value of all $\delta RPKM$ of a given gene would mean that this gene is "preferentially present" in the comparison samples group 2 than in the reference genome group 1.

2. calculates the **standard deviation** s of all the m_j values in the analysis

3. selects genes as present in the reference genome group 1 based on an **arbitrary threshold of** $2.s$:

$$selectedGene : m_j >= 2.s \tag{5}$$

   In other words, a gene j that presents a median $\delta RPKM$ value greater than $2.s$ will be considered as "preferentially present" in the reference genome group 1 (with <lineage_type> ="Lineage_type_I") than in the comparison group 2 (with <lineage_type> ="Lineage_type_II").

7.2 Run `deltaRpkm::deltaRPKMStats`

```
> stats_table <- deltaRPKMStats(deltarpkm_table = deltarpkm_table)
```

The default threshold value to select genes is based on $2.s$. But one can change this threshold in the `deltaRpkm::deltaRPKMStats` parameter `min_SD_foldChange`, *e.g*:

```
> stats_table_fcnew <- deltaRPKMStats(min_SD_foldChange = 1.5,
                                      deltarpkm_table = deltarpkm_table)
> head(stats_table_fcnew)
          geneID sample.group1 lineage_type.group1 infection.group1
             reads.group1 rpkm.group1 sample.group2 lineage_type.
             group2
1 LMJF5203_00001         JF4906            Lineage_I                CNS
             1177          425         JF4931           Lineage_II
2 LMJF5203_00001         JF4906            Lineage_I                CNS
             1177          425         JF5172           Lineage_II
3 LMJF5203_00001         JF4906            Lineage_I                CNS
             1177          425         JF5761           Lineage_II
  infection.group2 reads.group2 rpkm.group2 deltarpkm deltarpkm_median
      deltarpkm_medianSD thres_SD_median_value selected_gene
1                CNS         3109          433        -8                -31
                  114.24                228.48             -
2                CNS         1466          520        -95               -31
                  114.24                228.48             -
3                ENV         5582          465        -40               -31
                  114.24                228.48             -
```

Note the column **`selected_gene`** that contains information about whether a given gene should be selected as present preferentially in the reference genome group - noted as "+" - or not - noted as "-". These data column will be used later to filter the relevant genes.

7.3 Visual check of the m_j distribution

With the function `deltaRpkm::median_plot` one can check visually how all genes medians values of $\delta RPKM$ vary.

```
> median_plot(data_table = stats_table,
              gene_annotation_table = coverage_table_N51) # genes
                 annotation info
```

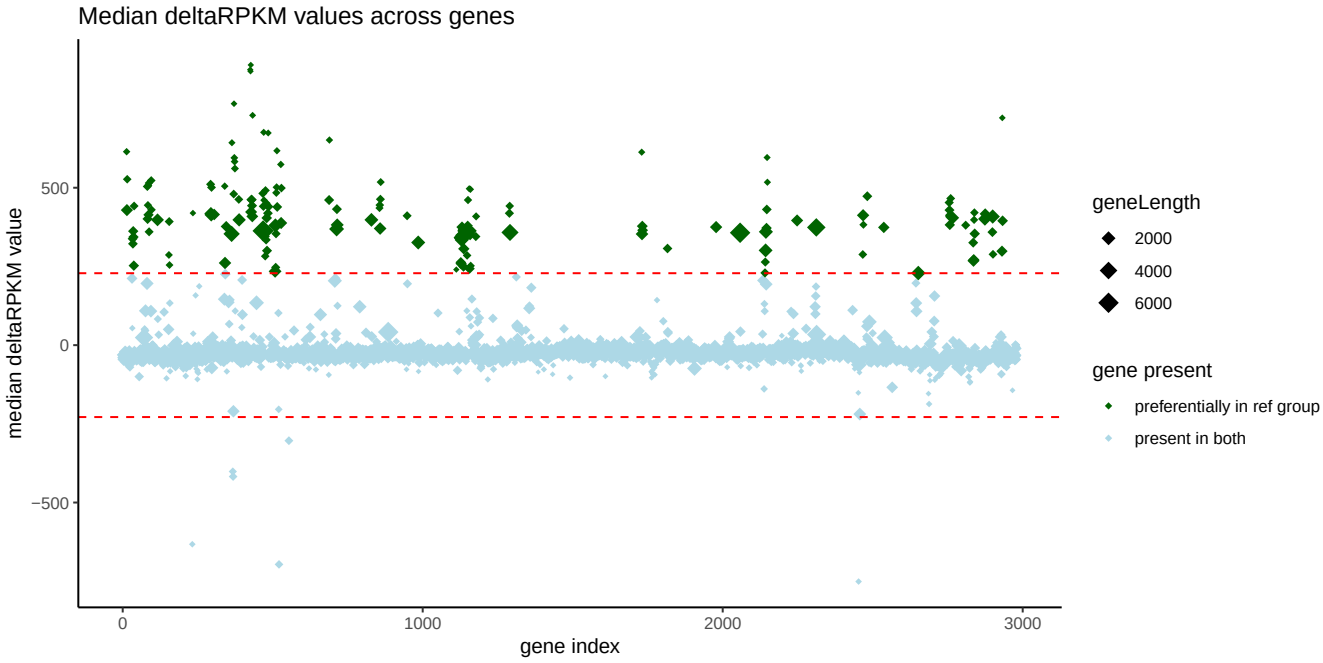


Figure 2: All genes median values of their $\delta RPKM$. Plot output from `deltaRpkm::median_plot`.The negative median $\delta RPKM$ values correspond to genes that appear as better covered in the comparison group 2 than in the reference genome's group 1. Note: the gene index value reflects the genomic coordinates since they are ordered as the gene names (these later being themselves given during *de novo* annotation based on the genomic coordinates, roughly speaking).

The genes in darkgreen in **Figure 2** correspond to the set of genes present in the reference genome group 1 and potentially linked to the studied phenotype ("lineage_type").

7.4 Selected genes set

For a given threshold value of median $\delta RPKM$ (by default $2.s$ of all median $\delta RPKM$ values), we can simply extract the genes appearing as differentially present in the reference genome group 1 (green dots in the **Figure 2**):

```
> differential_present_genes <- unique(stats_table[stats_table$
    selected_gene %in% "+", ]$geneID)
> length(differential_present_genes)
> [1] 173
> head(differential_present_genes)
> [1] "LMJF5203_00013" "LMJF5203_00014" "LMJF5203_00015" "LMJF5203
```

```
_00033" "LMJF5203_00034" "LMJF5203_00035"
```

This genes set can be used to perform various functional clustering analysis. We propose in the package a method to build a summary heatmap of their RPKM values and how they relate (or not) to a second phenotype of interest.

8 Heatmap

8.1 Rational

The idea is to analyse how the RPKM values of the genes specific to the reference genome group 1 distribute across all samples of group 1 and group 2. The aim is:

- to confirm (or infirm) that the heatmap clustering of the samples into two distinct categories is coherent with the initial group 1 and group 2 definition. Typically, the selected genes should present in overall higher RPKM values in the reference genome group 1 than in group 2
- to investigate at a higher resolution the homogeneity of each group

Thus deltaRpkm allows to investigate the clustering of the selected genes based on their RPKM values computed earlier. One can visualize a putative correlation with the second phenotype given in the metadata table - which is "infection" in the working example dataset.
The heatmap plot is made with the `deltaRpkm::rpkmHeatmap` function, derived from the `gplots::heatmap.2` method (Warnes et al., 2018).

8.2 Preparing the RPKM values for the heatmap

The heatmap will be focussing only on the *RPKM values of the set of genes that are relevant*, *i.e* the ones that appear as differentially present in the reference genome group 1 (see the dark-green dots in **Figure 2**).

For this, we first subset the RPKM data table and keep only the rows/genes that were selected using the `deltaRpkm::subsetRPKMTable`:

```
# Subset the RPKM table for the selected genes
> heatmap_table <- subsetRPKMTable(rpkm_table = rpkmtable,
                        user_metadata = design_table,
                        delta_phenotype_colname = "lineage_type",
                        heatmapbar_phenotype_colname = "infection",
                        sd_filtered_genes = differential_present_genes
                            )
> head(heatmap_table)
  sample lineage_type infection LMJF5203_00013 LMJF5203_00014 LMJF5203
      _00015 LMJF5203_00033 LMJF5203_00034 LMJF5203_00035 LMJF5203
      _00036
1 JF4906     Lineage_I       CNS            607            450
               630             421            445            521
               553
2 JF4929     Lineage_I       CNS            581            397
```

```
                498             456             447             427
                423
3 JF4931    Lineage_II         CNS               0               0
                  0              68               1               2
                  1
```

Then the subsetted RPKM values data frame is converted to a matrix (since this is the required format for the heatmap function) using the `deltaRpkm::convertHeatmapToMatrix` function:

```
# Convert the subsetted RPKM table to a matrix
> heatmap_matrix <- convertHeatmapToMatrix(wide_rpkm_table = heatmap_
    table,
                                delta_phenotype_colname = "lineage_type",
                                heatmapbar_phenotype_colname = "infection"
                                   )
> head(heatmap_matrix)
        LMJF5203_00013 LMJF5203_00014 LMJF5203_00015 LMJF5203_00033
           LMJF5203_00034 LMJF5203_00035 LMJF5203_00036 LMJF5203_00037
           LMJF5203_00038 LMJF5203_00082 LMJF5203_00083 LMJF5203_00084
JF4906              607             450             630             421
                  445             521             553             472
                  491             619             352             450
JF4929              581             397             498             456
                  447             427             423             412
                  484               0               0               0
JF4931                0               0               0              68
                    1               2               1               0
                    1               0              18             457
```

It is important to note that the heatmap matrix must contain samples names as row names.

8.3 Plot heatmap

Finally, we plot a summary plot as a heatmap to highlight the selected genes RPKMs difference between group 1 and group 2 samples, using the `deltaRpkm::rpkmHeatmap` function:

```
> rpkmHeatmap(filtered_rpkm_matrix = heatmap_matrix,
              user_metadata = design_table,
              heatmapbar_phenotype_colname = "infection")
```

This creates an output heatmap file - `deltaRpkm_heatmap.tiff` - in the working directory. The heatmap for the example dataset (*Listeria monocytogenes, N=51*) is shown in **Figure 3**. It confirms the clustering of the samples into the initial two categories: the group 1 samples cluster together on the upper part corresponding to high RPKM values, while the group 2 samples cluster together in the lower part of the heatmap with lower RPKM values, for the selected genes set.
The heatmap colors can be easily changed with the color breaks parameters:

```
> rpkmHeatmap(filtered_rpkm_matrix = heatmap_matrix,
              user_metadata = design_table,
              heatmapbar_phenotype_colname = "infection",
              low_color = "col1",  # low RPKM values, default "yellow"
              mid_color = "col2",  # mid range RPKM values, default "
                  green"
              high_color = "col3") # high range RPKM values, default "
                  blue"
```

See next section for more on color breaks tuning.

8.4 Tuning heatmap parameters: color breaks

Note that `deltaRpkm::rpkmHeatmap` comes with various parameters options (see `?rpkmHeatmap`), some derived from the original `gplots::heatmap.2`, and some specific to deltaRpkm analysis.
In particular, the heatmap **color breaks** can be adjusted with i) the `binsize` (default 200), ii) the `lower_limit` (default value 300) and iii) `upper_limit` (default value 550) arguments. These values are based on the RPKM values distribution and correspond to the lower and upper boundary RPKM values of the main peak:

```
> hist(rpkmtable$rpkm, freq = FALSE, breaks = 1000)
```

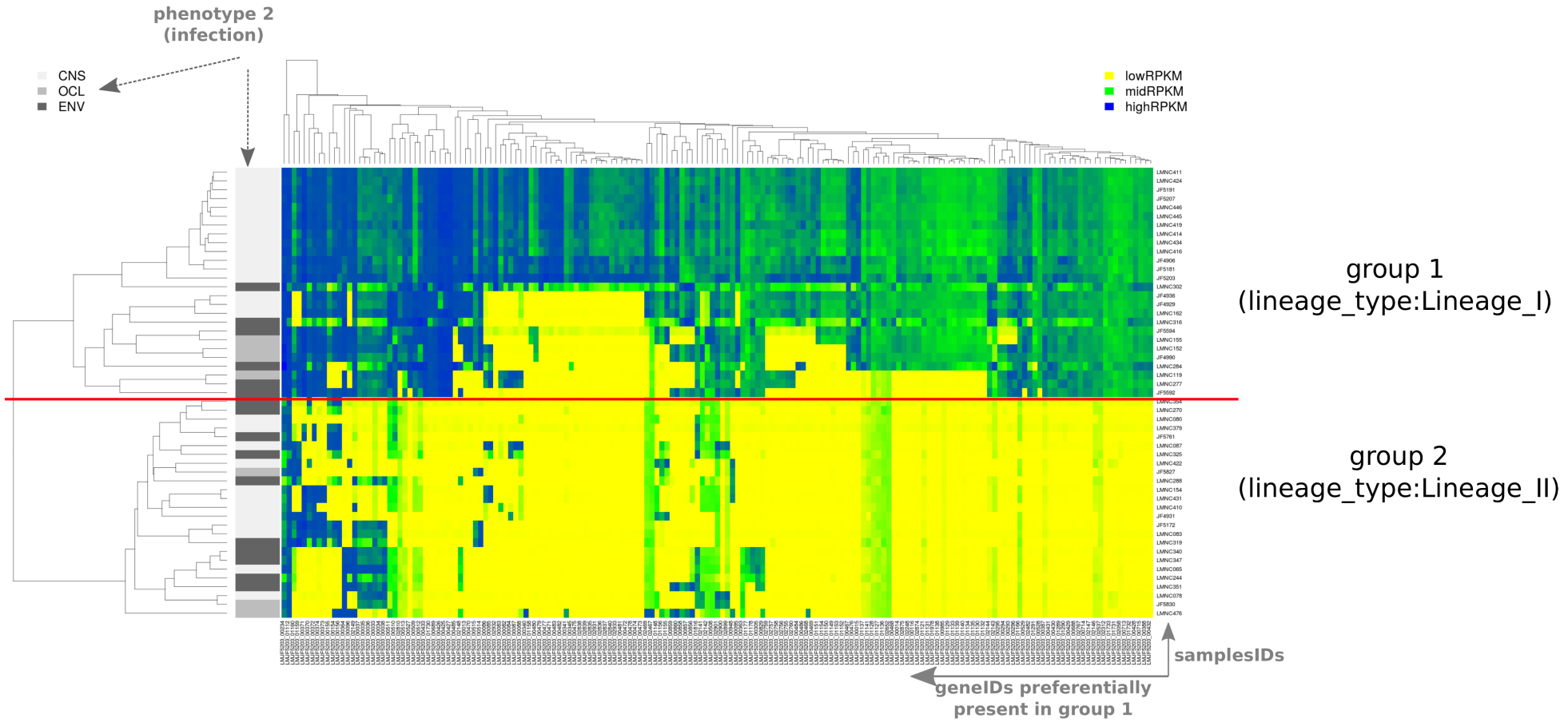


Figure 3: Selected genes RPKMs distribution across samples from group 1 and group 2. Plot output from `deltaRpkm::rpkmHeatmap`. The samples cluster following the phenotype 1 ("lineage_type"), with group 1 annotated as "Lineage_I" and group 2 annotated as "Lineage_II". Most of the selected genes sets appear with a low RPKM value in group 2 cluster, eventhough some group 2 samples present high RPKM values (blue pixels) for certain geneIDs, suggesting these as potential false positives.

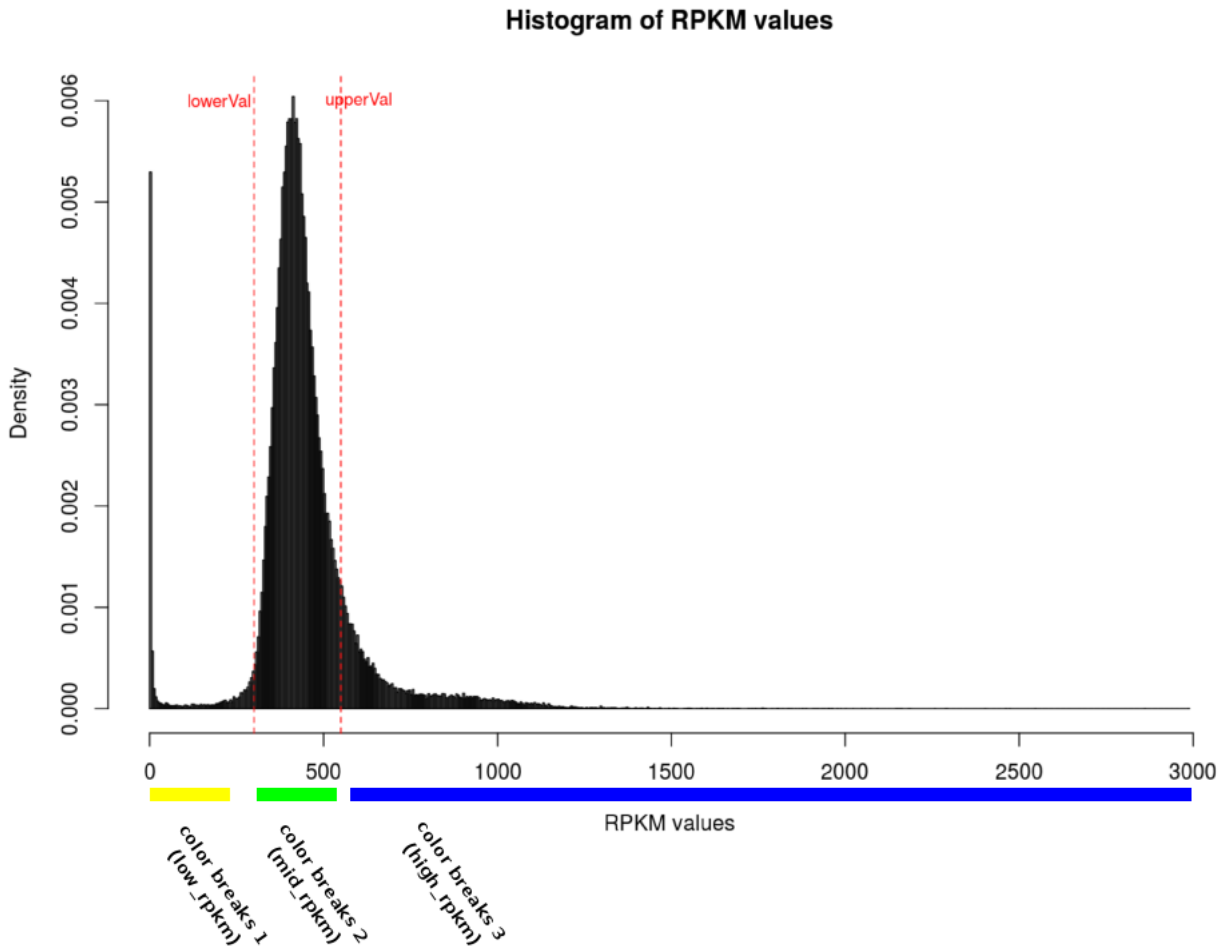


Figure 4: Distribution of RPKM values: inferring the heatmap color breaks from the histogram main peak boundary values. The lower (∼300) and upper (∼550) values of RPKM are used in the `deltaRpkm::rpkmHeatmap` to adjust the heatmap color breaks. Working dataset *Listeria monocytogenes, N=51*.

Also, deltaRPKM proposes some methods to infer these RPKM boundary values with the function `deltaRpkm::boundaries`:

```
# default method mclust
> res <- boundaries(x = rpkmtable$rpkm)
> res$boundaries_df
    boundaries rpkm_values
1 lower_limit          300
2 upper_limit          624

> res <- boundaries(x = rpkmtable$rpkm, strategy = "ratios")
>     boundaries rpkm_values
1 lower_limit          295
2 upper_limit          585

> res <- boundaries(x = rpkmtable$rpkm, strategy = "quartiles")
>     boundaries rpkm_values
    boundaries rpkm_values
1 lower_limit          383
2 upper_limit          487
```

deltaRpkm::boundaries applies by default the **mclust**, which is derived from the method mclust::densityMclust. This can be changed with the parameter strategy. The boundary RPKM values can be simply extracted as res$boundaries_df containing the RPKM boundary values of interest.

Feel free to play with these RPKM boundary and colors breaks parameters values in rpkmHeatmap function and observe the effect(s) on the heatmap readout.

9 Check and correct potential batch effect (optional)

It has been assumed so far that the datasets were not biased. It is up to the user to ensure the quality of its input dataset and the absence of potential biases. Nevertheless, we strongly recommend to investigate for potential batch effect. In case of batch effect, the RPKM values should be corrected accordingly and the deltaRpkm analysis re-run with the new values.
We propose below a method to check and correct batch effects in the example working datasets that have been used.

The deltaRpkm package includes the sva::Combat function from sva package in order to check if there is any bias and correct the RPKM values accordingly. An "all-in-one method" is included in deltaRpkm::batchCorrectRpkm, which allows to compute an rpkm table with corrected values and PCA plots - before and after correction - to check the potential effect of the correction.
Here we suspect that the sequencing platform - HiSeq ∼ MiSeq - might induce a bias in the RPKM values. We apply the batch effect correction based on the <platform> column by giving it in the **batch_colname** argument:

```
corr_rpkmtable <- batchCorrectRpkm(batch_colname = "platform",
                        batch_info_table = metadata_table_N51,
                        rpkm_table = rpkmtable,
                        sample_colname = "sample",
                        delta_phenotype_colname = "lineage_type",
                        heatmapbar_phenotype_colname = "infection")
```

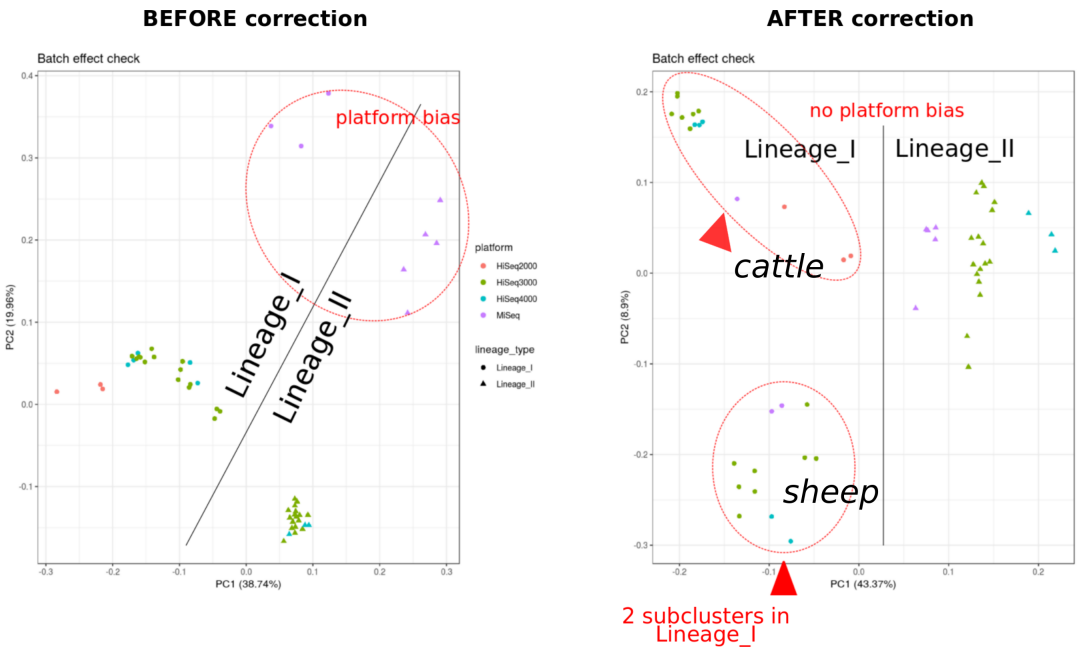


Figure 5: PCA: RPKM values are biased by the "platform" factor. The PCA on the original RPKM values show a potential bias in the Lineage_type clustering that could be due to the sequencing technology (plot BEFORE). By taking into account the platform factor, the bias is corrected (AFTER). A closer analysis of the PCA with corrected RPKM shows a subclustering of the Lineage_I group into 2 subsets, corresponding to "Cattle" and "Sheep". Working dataset *Listeria monocytogenes*, N=51.

10 Re-run the analysis with corrected RPKM values

10.1 corrRPKM, $\delta corrRPKM$

```
# correct the RPKM values
corr_rpkmtable <- batchCorrectRpkm(batch_colname = "platform",
                                   batch_info_table = metadata_table
                                       _N51,
                                   rpkm_table = rpkmtable,
                                   sample_colname = "sample",
                                   delta_phenotype_colname = "
                                       lineage_type",
                                   heatmapbar_phenotype_colname = "
                                       infection",
                                   plot_labels = FALSE)


# run deltarpkm function
corr_deltarpkm_table <- deltarpkm(rpkm_table = corr_rpkmtable,
                                  genes_names = unique(corr_rpkmtable$
                                      geneID),
                                  samples_colname = "sample",
                                  delta_phenotype_colname = "lineage_type",
                                  reference_sample = "JF5203",
                                  nonref_delta_phenotype = "Lineage_II")

# compute medians, standard deviations, selection threshold:
```

```
corr_stats_table <- deltaRPKMStats(deltarpkm_table = corr_deltarpkm_
    table)  # default value already included
```

10.2 Median $\delta corrRPKM$

```
# plot medians values distribution
median_plot(data_table = corr_stats_table,
            gene_annotation_table = coverage_table_N51[, c("start", "
               end", "geneID")])
```

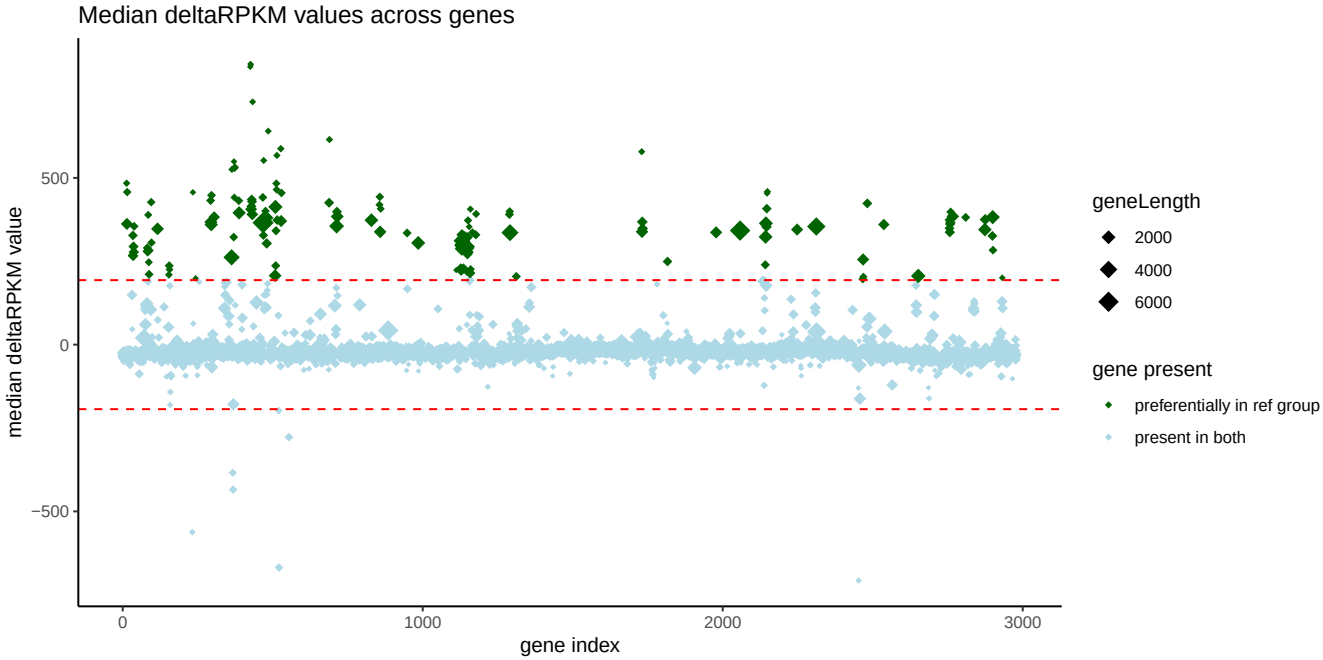


Figure 6: Median $\delta corrRPKM$ values distribution plot. With corrected RPKM values.

10.3 Gene filtering

```
# selected genes
corr_differential_present_genes <- unique(corr_stats_table[corr_stats_
    table$selected_gene %in% "+", ]$geneID)
```

The comparison between the set of genes that are preferentially present in the reference genome group 1, before (N=173) and after correction (N=150) of the RPKM values indicates that most of the geneIDs are conserved after correction (N=148), indicating that in this working example the batch effect due to the "platform" factor did not have a strong incidence on the overall outcome:

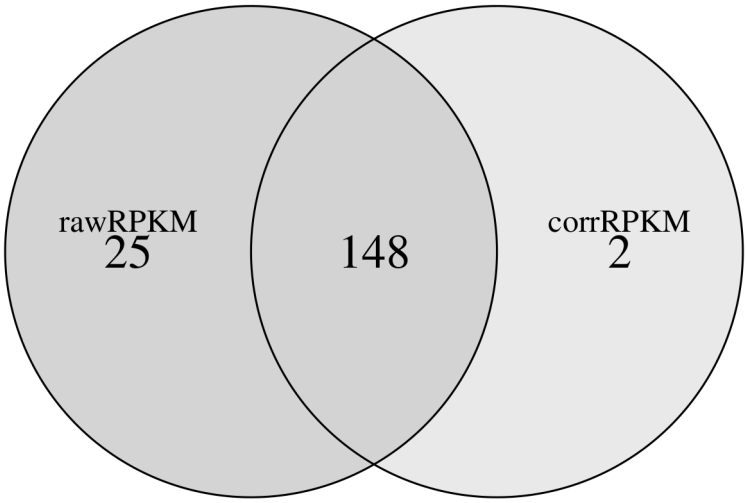


Figure 7: Venn Diagram of selected genes, before and after RPKM correction for platform batch effect. The batch effect was not too severe with this present dataset.

10.4 Heatmap with corrected RPKM values

```
# Convert the subset RPKM table to a matrix
corr_heatmap_matrix <- convertHeatmapToMatrix(wide_rpkm_table = corr_
    heatmap_table,
                                              delta_phenotype_colname = "
                                                 lineage_type",
                                              heatmapbar_phenotype_colname
                                                 = "infection")

# Heatmap plot
rpkmHeatmap(filtered_rpkm_matrix = corr_heatmap_matrix,
            user_metadata = design_table,
            heatmapbar_phenotype_colname = "lineage_type")
```

The output plot is shown in **Figure 8**.

Overall, with the example working dataset used in these guidelines, the batch effect observed with the sequencing platform did not affect "too much" the output results. But it is a good practice to always check for potential bias in the input datasets and correct the RPKM accordingly before running the deltaRpkm pipeline.

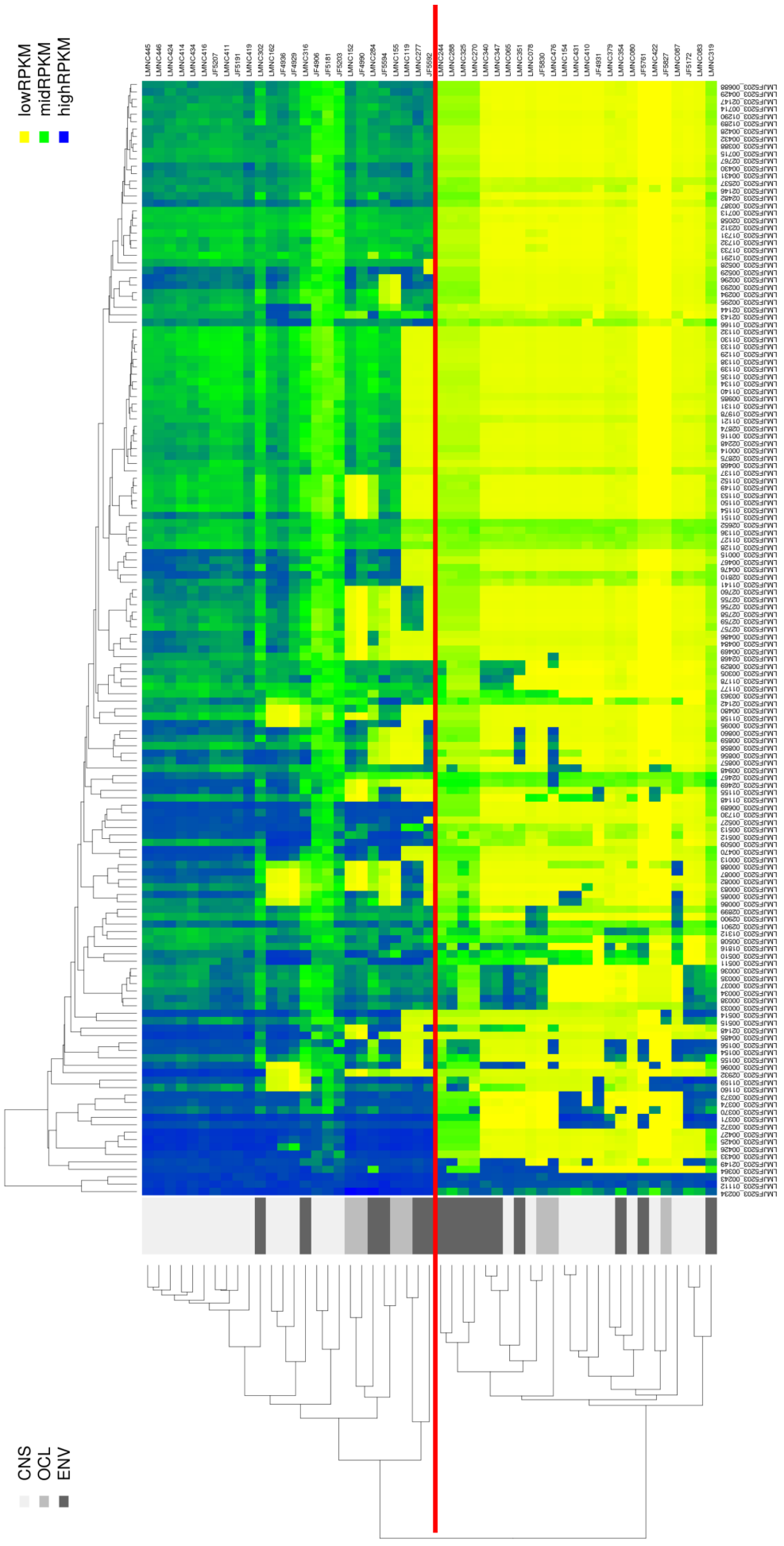


Figure 8: Heatmap of selected genes with corrected RPKM values.

11 deltaRpkm performance: downsampling

The initial *Listeria monocytogenes* dataset of N=225 samples is downsampled up to N=7 samples. Each dataset is run through deltaRpkm pipeline and the different outcomes are compared.

11.1 Dataset size effect on thresholding and gene set selection

The gene differential presence is based on a threshold value defined as 2 times (default value) the standard deviation of the medians of $\delta RPKM$ values. The median plots (see **Figure 2**) for all the datasets of different sizes can be summarized in a single boxplot, as shown in **Figure 9**.

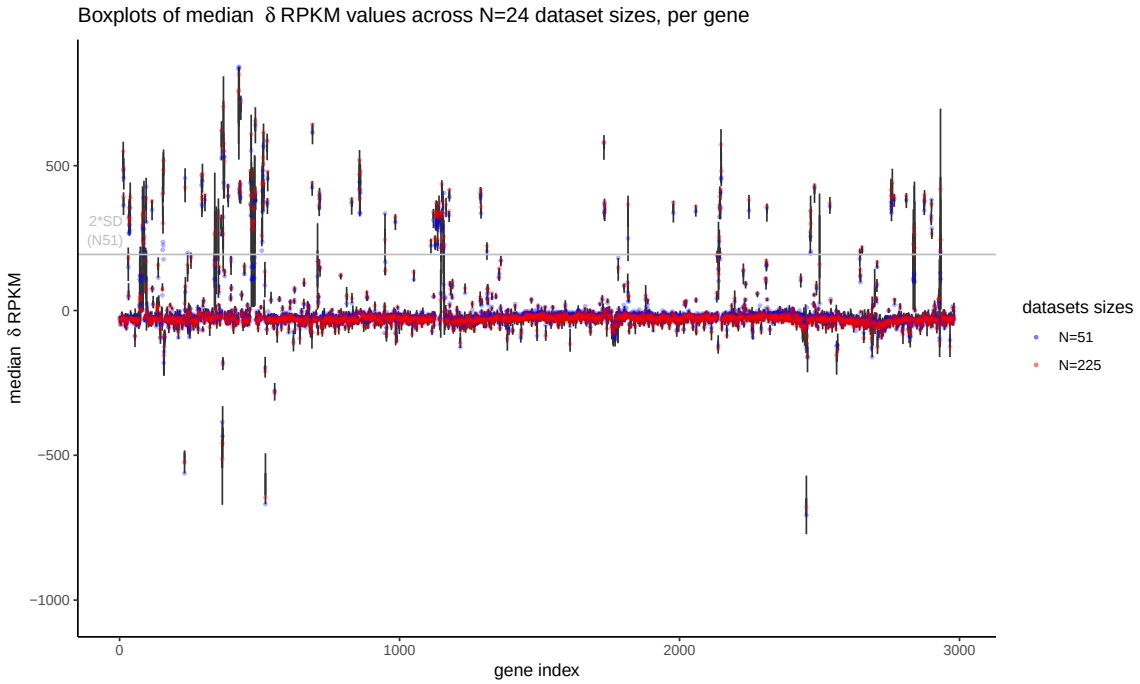


Figure 9: Dataset size effect on the median $\delta RPKM$ distribution (boxplots series). N=24 datasets of different sizes (N=225 to N=7 samples) are plotted as boxplots, one per gene. Datasets with N=51 and N=225 samples are highlighted, showing that the median $\delta RPKM$ values of a given geneID (m_j) does not vary so much between datasets of different sizes.

We can see in the boxplot series of **Figure 9** that most of the genes that are selected present a consistent median $\delta RPKM$ distribution across all datasets sizes (bars fully above the $2.s$ threshold).

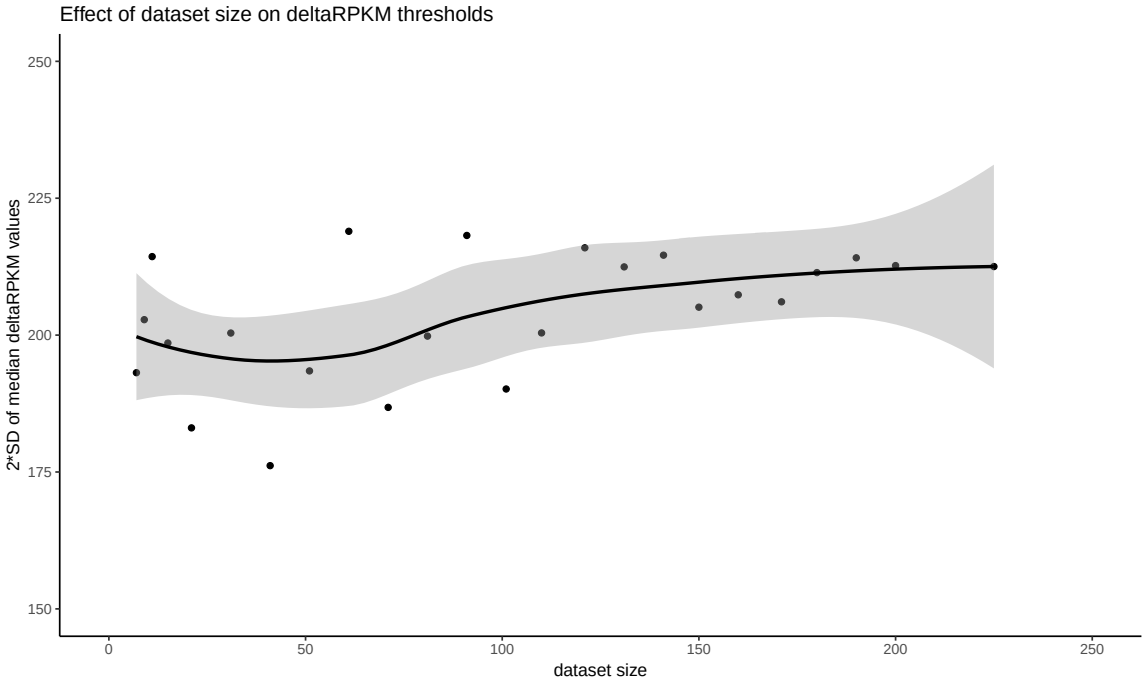


Figure 10: Dataset size effect on the 2*SD(median $\delta RPKM$) thresholding values.

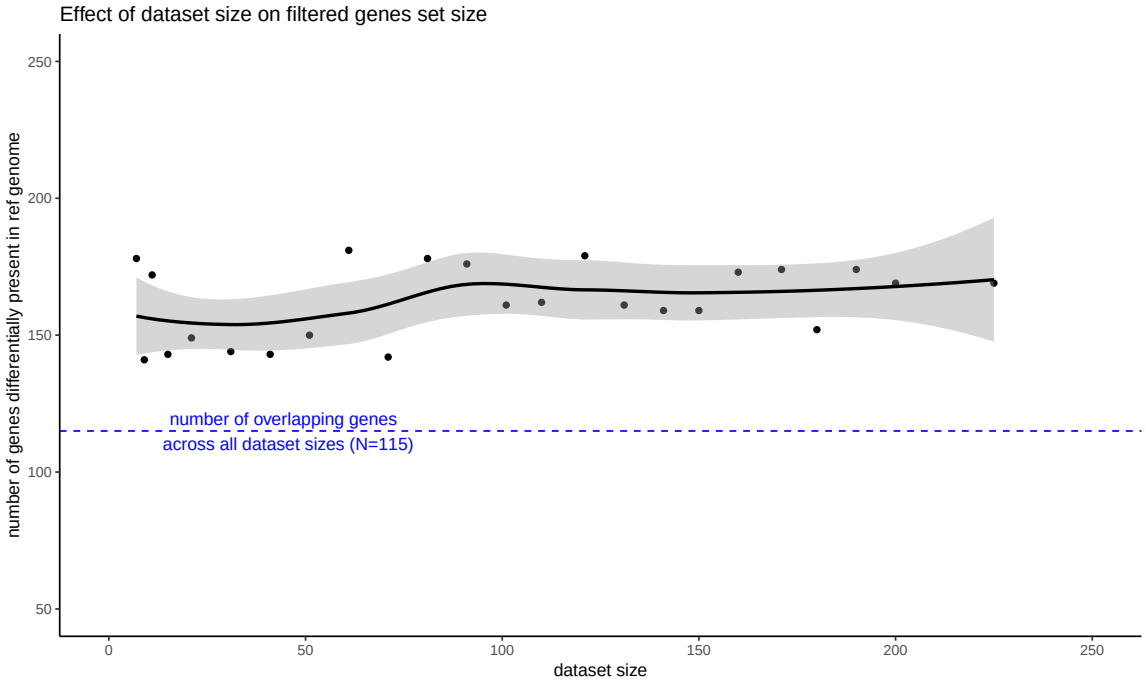


Figure 11: Dataset size effect on genes differentially present in reference genome.

11.2 Dataset size effect on runtime

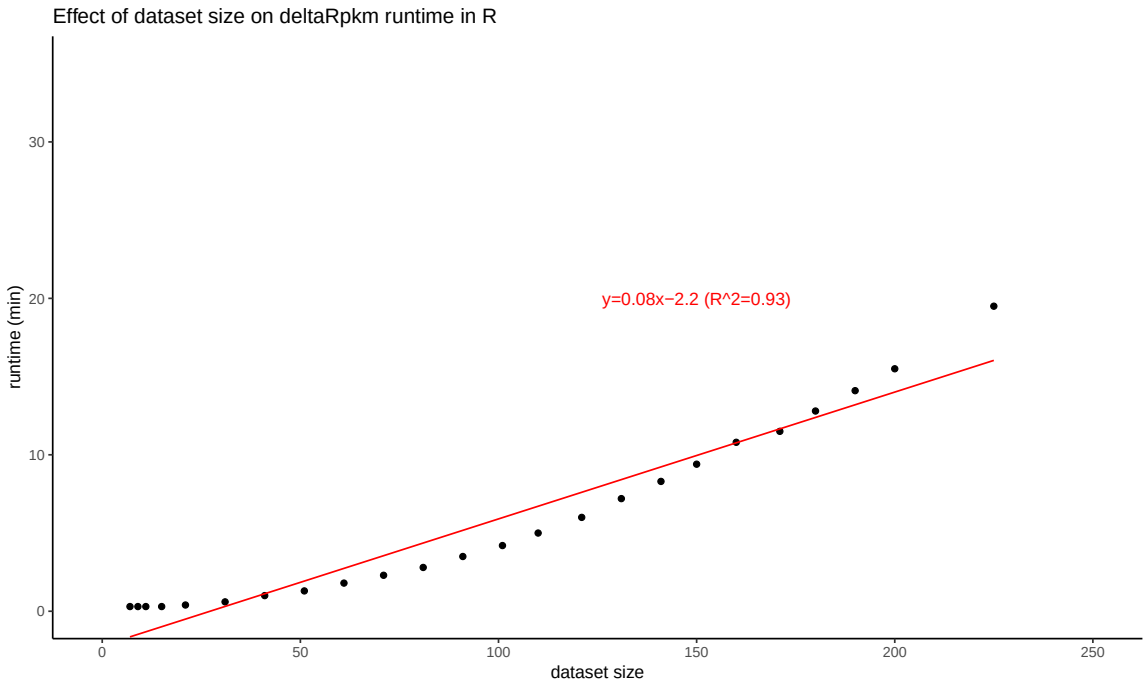


Figure 12: Dataset size and runtime when using deltaRpkm. Including the pipeline steps of RPKM computing, batch effect correction, computing $\delta RPKM$ values, statistics, gene selection and plotting heatmap. Analysing with deltaRpkm a large dataset of N=225 samples takes $<$ **20min** in total in R 3.4.4 (under Ubuntu 14.04.5 LTS).

11.3 Dataset size effect on memory usage

The memory requirement by deltaRpkm analysis grows with the sample size, but in a rather linear way: expect ∼400M every N∼20 samples. So one should be able to run a dataset of up to N∼800 samples on a normal desktop machine with 16G of RAM.

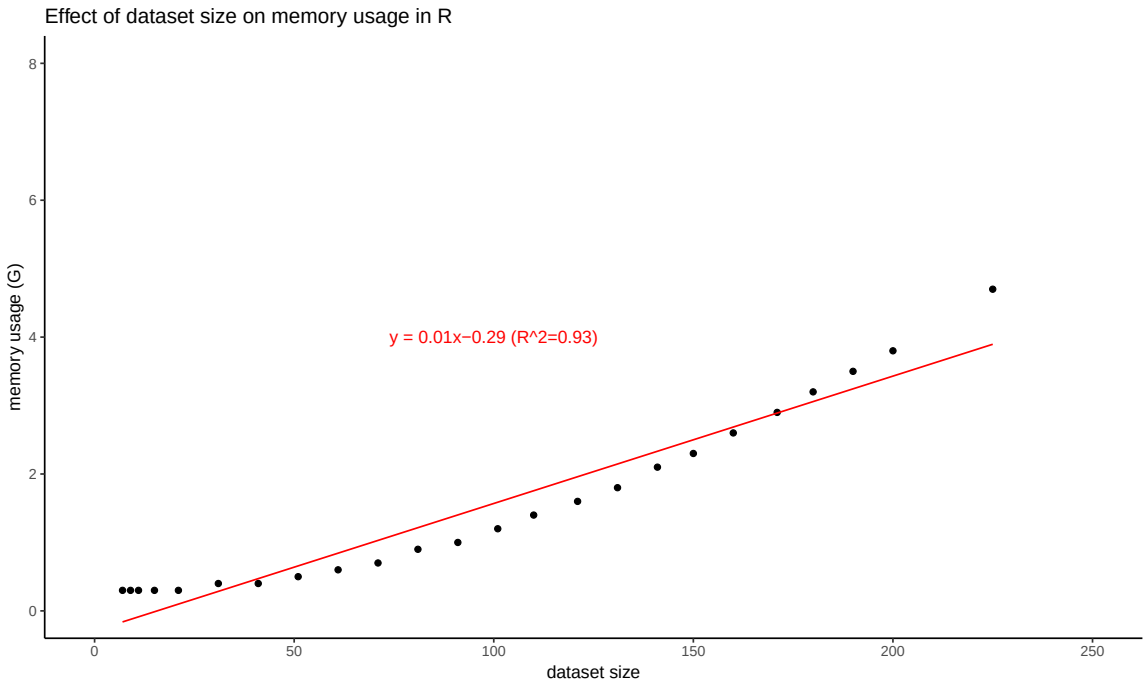


Figure 13: Dataset size and memory usage when using deltaRpkm. The pipeline uses $<$ **4G of memory** for a dataset of N=225 samples, when ran in R 3.4.4 (under Ubuntu 14.04.5 LTS).

11.4 Random datasets

Random datasets confirm the robustness of selected genes with the deltaRpkm method (**Fig 14**). When comparing datasets of different sizes (N=51, N=101, N=225) containing either real or random samples groups, most of genes that are identified as differentially present in the group 1 are conserved across datasets (**Fig 14(a)**, N=144). While on the other hand, the genes sets derived from random groupings are small and not consistent (**Fig 14(b)**, N=0).

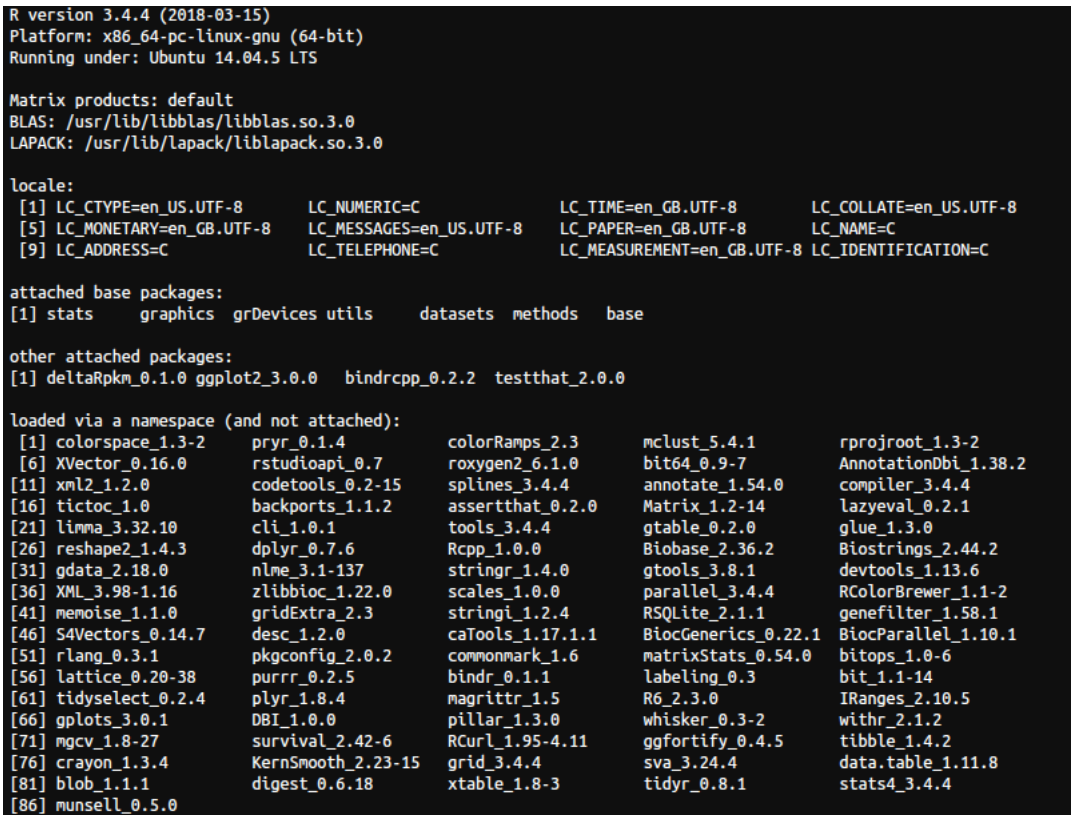
(a) real datasets **(b)** random datasets

Figure 14: Real versus random datasets: random datasets give non-robust gene sets across different sizes of datasets. deltaRpkm *Listeria monocytogenes* datasets. Batch effect corrected.

12 Binaries and OS platforms

12.1 Ubuntu Trusty Tahr (14.04.5 LTS)

The Linux binary has been built and tested with `R 3.4` under `Ubuntu 14.04 LTS (Trusty Tahr)`:



```
R version 3.4.4 (2018-03-15)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.5 LTS

Matrix products: default
BLAS: /usr/lib/libblas/libblas.so.3.0
LAPACK: /usr/lib/lapack/liblapack.so.3.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_GB.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_US.UTF-8    LC_PAPER=en_GB.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C             LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] deltaRpkm_0.1.0 ggplot2_3.0.0   bindrcpp_0.2.2  testthat_2.0.0

loaded via a namespace (and not attached):
 [1] colorspace_1.3-2     pryr_0.1.4           colorRamps_2.3       mclust_5.4.1         rprojroot_1.3-2
 [6] XVector_0.16.0       rstudioapi_0.7       roxygen2_6.1.0       bit64_0.9-7          AnnotationDbi_1.38.2
[11] xml2_1.2.0           codetools_0.2-15     splines_3.4.4        annotate_1.54.0      compiler_3.4.4
[16] tictoc_1.0           backports_1.1.2      assertthat_0.2.0     Matrix_1.2-14        lazyeval_0.2.1
[21] limma_3.32.10        cli_1.0.1            tools_3.4.4          gtable_0.2.0         glue_1.3.0
[26] reshape2_1.4.3       dplyr_0.7.6          Rcpp_1.0.0           Biobase_2.36.2       Biostrings_2.44.2
[31] gdata_2.18.0         nlme_3.1-137         stringr_1.4.0        gtools_3.8.1         devtools_1.13.6
[36] XML_3.98-1.16        zlibbioc_1.22.0      scales_1.0.0         parallel_3.4.4       RColorBrewer_1.1-2
[41] memoise_1.1.0        gridExtra_2.3        stringi_1.2.4        RSQLite_2.1.1        genefilter_1.58.1
[46] S4Vectors_0.14.7     desc_1.2.0           caTools_1.17.1.1     BiocGenerics_0.22.1  BiocParallel_1.10.1
[51] rlang_0.3.1          pkgconfig_2.0.2      commonmark_1.6       matrixStats_0.54.0   bitops_1.0-6
[56] lattice_0.20-38      purrr_0.2.5          bindr_0.1.1          labeling_0.3         bit_1.1-14
[61] tidyselect_0.2.4     plyr_1.8.4           magrittr_1.5         R6_2.3.0             IRanges_2.10.5
[66] gplots_3.0.1         DBI_1.0.0            pillar_1.3.0         whisker_0.3-2        withr_2.1.2
[71] mgcv_1.8-27          survival_2.42-6      RCurl_1.95-4.11      ggfortify_0.4.5      tibble_1.4.2
[76] crayon_1.3.4         KernSmooth_2.23-15   grid_3.4.4           sva_3.24.4           data.table_1.11.8
[81] blob_1.1.1           digest_0.6.18        xtable_1.8-3         tidyr_0.8.1          stats4_3.4.4
[86] munsell_0.5.0
```

Figure 15: Session info in R 3.4.4, with RStudio 1.0.143, run under Ubuntu 14.04.5 LTS.

12.2 Ubuntu (18.04.2 LTS)

The Linux binary has been built and tested with R 3.4 under Ubuntu 18.04.2 LTS (Bionic Beaver):

Figure 16: Session info in R 3.4.4, with RStudio 1.1.463, run under Ubuntu 18.04.2 LTS.

12.3 MacOS High Sierra (10.13.6)

The MacOS binary has been built and tested with R 3.4 under MacOS 10.13.6 (High Sierra):

```
R version 3.4.0 (2017-04-21)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS  10.13.6

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A/libBLAS.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ggfortify_0.4.5 ggplot2_3.1.0   deltaRpkm_0.1.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.0          lattice_0.20-38     tidyr_0.8.2         gtools_3.8.1
 [5] assertthat_0.2.0    digest_0.6.18       R6_2.4.0            plyr_1.8.4
 [9] stats4_3.4.0        RSQLite_2.1.1       sva_3.26.0          pillar_1.3.1
[13] gplots_3.0.1.1      rlang_0.3.1         lazyeval_0.2.1      gdata_2.18.0
[17] rstudioapi_0.9.0    annotate_1.56.2     blob_1.1.1          S4Vectors_0.16.0
[21] Matrix_1.2-15       labeling_0.3        splines_3.4.0       BiocParallel_1.12.0
[25] stringr_1.4.0       RCurl_1.95-4.11     bit_1.1-14          munsell_0.5.0
[29] compiler_3.4.0      pkgconfig_2.0.2     BiocGenerics_0.24.0 mgcv_1.8-27
[33] tidyselect_0.2.5    tibble_2.0.1        gridExtra_2.3       IRanges_2.12.0
[37] matrixStats_0.54.0  XML_3.98-1.17       crayon_1.3.4        dplyr_0.8.0.1
[41] withr_2.1.2         bitops_1.0-6        grid_3.4.0          nlme_3.1-137
[45] xtable_1.8-3        gtable_0.2.0        DBI_1.0.0           magrittr_1.5
[49] scales_1.0.0        KernSmooth_2.23-15  stringi_1.3.1       reshape2_1.4.3
[53] genefilter_1.60.0   limma_3.34.9        RColorBrewer_1.1-2  tools_3.4.0
[57] bit64_0.9-7         Biobase_2.38.0      glue_1.3.0          purrr_0.3.0
[61] parallel_3.4.0      survival_2.43-3     AnnotationDbi_1.40.0 colorspace_1.4-0
[65] caTools_1.17.1.1    memoise_1.1.0
```

Figure 17: Session info in R 3.4.0 with RStudio, run under MacOS 10.13.6.

12.4 Windows7

The Windows binary has been built and tested with `R 3.5` under `Windows7`:

```
R version 3.5.2 (2018-12-20)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 7 x64 (build 7601) Service Pack 1

Matrix products: default

locale:
[1] LC_COLLATE=French_Switzerland.1252  LC_CTYPE=French_Switzerland.1252    LC_MONETARY=French_Switzerland.1252
[4] LC_NUMERIC=C                        LC_TIME=French_Switzerland.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] deltaRpkm_0.1.0 ggfortify_0.4.5 ggplot2_3.1.0

loaded via a namespace (and not attached):
 [1] Biobase_2.42.0       pkgload_1.0.2        tidyr_0.8.2          bit64_0.9-7          splines_3.5.2
 [6] gtools_3.8.1         assertthat_0.2.0     BiocManager_1.30.4   stats4_3.5.2         blob_1.1.1
[11] remotes_2.0.2        sessioninfo_1.1.1    pillar_1.3.1         RSQLite_2.1.1        backports_1.1.3
[16] lattice_0.20-38      glue_1.3.0           limma_3.38.3         digest_0.6.18        RColorBrewer_1.1-2
[21] colorspace_1.4-0     Matrix_1.2-15        plyr_1.8.4           XML_3.98-1.17        pkgconfig_2.0.2
[26] devtools_2.0.1       genefilter_1.64.0    purrr_0.3.0          xtable_1.8-3         scales_1.0.0
[31] gdata_2.18.0         processx_3.2.1       BiocParallel_1.16.6  tibble_2.0.1         annotate_1.60.0
[36] mgcv_1.8-27          IRanges_2.16.0       usethis_1.4.0        withr_2.1.2          BiocGenerics_0.28.0
[41] lazyeval_0.2.1       cli_1.0.1            survival_2.43-3      magrittr_1.5         crayon_1.3.4
[46] memoise_1.1.0        ps_1.3.0             fs_1.2.6             nlme_3.1-137         gplots_3.0.1.1
[51] pkgbuild_1.0.2       tools_3.5.2          prettyunits_1.0.2    matrixStats_0.54.0   stringr_1.4.0
[56] S4Vectors_0.20.1     munsell_0.5.0        AnnotationDbi_1.44.0 callr_3.1.1          compiler_3.5.2
[61] caTools_1.17.1.1     rlang_0.3.1          grid_3.5.2           RCurl_1.95-4.11      rstudioapi_0.9.0
[66] bitops_1.0-6         labeling_0.3         gtable_0.2.0         DBI_1.0.0            reshape2_1.4.3
[71] R6_2.4.0             gridExtra_2.3        dplyr_0.8.0.1        bit_1.1-14           rprojroot_1.3-2
[76] KernSmooth_2.23-15   desc_1.2.0           stringi_1.3.1        parallel_3.5.2       sva_3.30.1
[81] Rcpp_1.0.0           tidyselect_0.2.5
```

Figure 18: Session info in R 3.5.2, with RStudio 1.1.463, run under Windows7.