

Glimma: Interactive Graphics for RNA-seq Analyses

User's Guide

Shian Su, Charity W. Law, Matthew E. Ritchie

First edition 28 January 2016
Last revised 27 March 2020

Contents

1	Quick start	3
2	Creating and sharing output	7
3	Multi-dimensional scaling plots	8
4	Mean-difference plots	10
4.1	General	10
4.2	Plotting options	11
4.3	Table options	12
5	XY plots	14
6	Using microarray data	16
7	Appendix	21
7.1	Extra mean-difference plots	21
7.1.1	edgeR-style analysis	21
7.1.2	DESeq2-style analysis	21
7.2	R session information	21

1 Quick start

Glimma is a Bioconductor [11] package for interactive visualization of results from differential expression analyses of RNA-sequencing (RNA-seq) and microarray data. Its functionality is intended to enhance reporting capabilities so that results can be explored more conveniently by end-users. Glimma, which loosely stands for interactive **G**raphics from **limma**, extends some of the popular plotting capabilities in the limma [1] package such as multi-dimensional scaling (MDS) plots and mean-difference (MD) plots. For seamless integration between the analysis by external packages and Glimma's interactive plots, Glimma accepts differential expression results from limma, edgeR [3] or DESeq2 [4] packages as input and creates an html page which presents the results interactively. Figure 1 gives an overview of the input data types and processing functions in Glimma. The displays within Glimma were inspired by visualisations from Degust software [2].

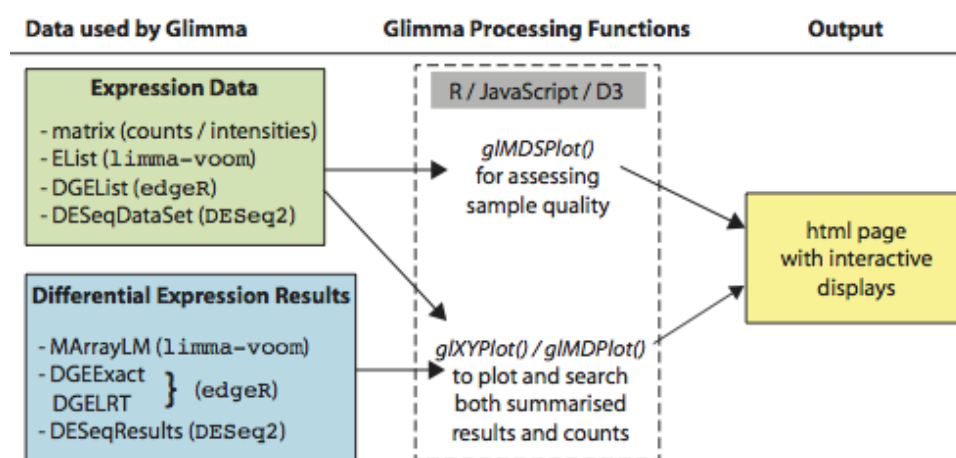


Figure 1: Overview of workflow showing the input and output types for functions in Glimma.

The main dataset used in this vignette is taken from an RNA-seq experiment examining lymphoma cell lines in mice with alterations to the *Smchd1* gene [5]. The count data is available as an edgeR DGEList object within Glimma for 4 wildtype samples and 3 samples that have a null allele of the *Smchd1* gene (we call these samples *Smchd1-null*).

```
library(Glimma)
library(limma)
library(edgeR)
data(lymphomaRNAseq)
rnaseq <- lymphomaRNAseq
rnaseq$samples$group
```

```
## [1] Smchd1-null Smchd1-null Smchd1-null Smchd1-null WT
## Levels: WT Smchd1-null
```

WT

WT

Lowly expressed genes are removed from downstream analysis and TMM-normalisation [9] is carried out.

```
rnaseq <- rnaseq[rowSums(cpm(rnaseq)>1)>=3,]
rnaseq <- calcNormFactors(rnaseq)
```

Using the `gIMDSPlot` function, an interactive MDS plot can be created to examine the clustering of samples in an unsupervised fashion. Distances in the plot represent similarities and dissimilarities between samples. Glimma's MDS plot allows users to interactively browse through different dimensions of the plot. A MDS plot is created here using a `DGEList` object of sample expression and vector specifying sample groups, a screen-capture of the html output is shown in Figure 2.

```
groups <- rnaseq$samples$group
gIMDSPlot(rnaseq, groups=groups)
```

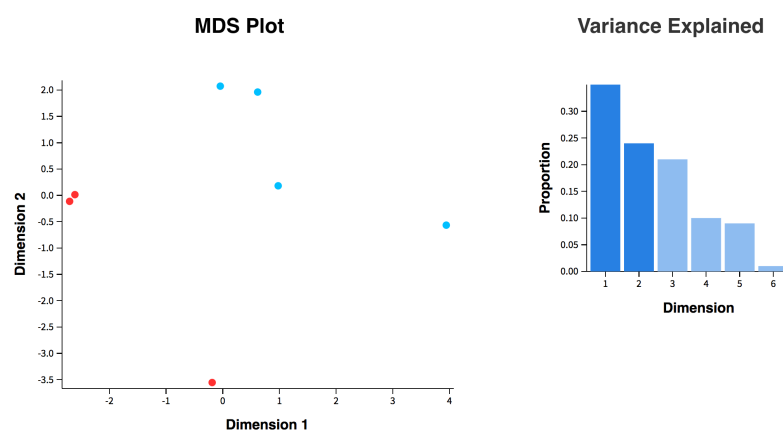


Figure 2: Interactive MDS plot where the dimensions displayed in the MDS plot (left) can be changed by clicking on the associated bars in the barplot (right). Samples, or points, in the MDS plot are colored by genotype.

We demonstrate the usage of Glimma by carrying out a limma-style analysis and using the corresponding output as input to Glimma functions. The same functions would work just as easily on output from edgeR or DESeq2 analyses, where examples of these are shown explicitly in the Appendix in Section 7. Here, differential expression of genes between *Smchd1*-null and wildtype samples is carried out using limma's voom with quality weights method [6, 7]. An adjusted p-value cutoff of 5% detects 882 genes as down-regulated in the *Smchd1*-null group relative to wildtypes, and 634 genes as up-regulated.

```
design <- model.matrix(~0+groups)
contrasts <- cbind(Smchd1null.vs.WT=c(-1,1))
vm <- voomWithQualityWeights(rnaseq, design=design)
fit <- lmFit(vm, design=design)
fit <- contrasts.fit(fit, contrasts)
fit <- eBayes(fit)
dt <- decideTests(fit)
summary(dt)
```

```
##          Smchd1null.vs.WT
## Down          879
## NotSig       10083
## Up           633
```

Glimma's interactive MD plot displays gene-wise log₂-fold changes (logFCs) against average expression values together with a plot of sample expression. This allows users to see summarised results from all of the genes as a whole whilst being able to scrutinise the expression of individual genes at the same time. Using the `gIMDPlot` function, a MD plot is created using `fit` which is an `MArrayLM` object, and `dt` a `TestResults` object that is used to highlight differentially expressed (DE) genes. The `EList` object from `voom` contains log₂-counts per million (logCPM) values which are used in the plot of sample expression. A screen-capture of the html output is shown in Figure 3.

```
gIMDPlot(fit, status=dt, counts=vm, groups=groups, side.main="Symbols")
```

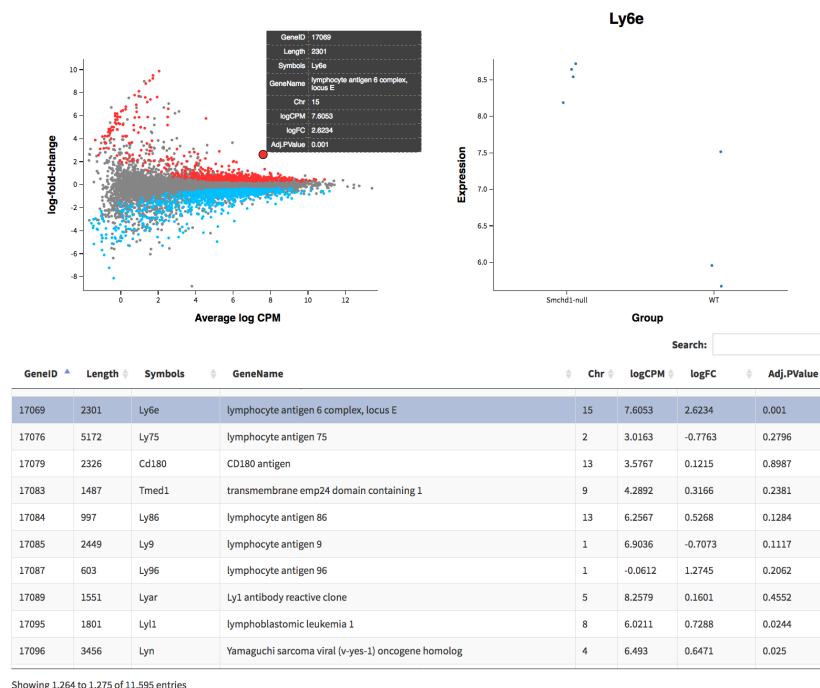


Figure 3: Interactive MD plot where gene-wise logFCs are plotted against mean expression values (top left). Significantly up- and down-regulated genes are highlighted in red and blue respectively. A table of associated gene information is displayed (bottom). Sample expression is displayed for an given gene (top right) by selecting a point in the main plot or a row in the table.

For a general plot of any two gene-wise summarised statistics, the `gIXYPlot` allows one to plot any two vectors of equal length against each other and associate these with sample expression. We display below the R-code for creating a volcano plot using logFC values and log-odds. It is important to ensure that ordering of genes is the same for the two vectors and the expression matrix!

```
glXYPLOT(x=fit$coef, y=fit$lod, xlab="logFC", ylab="logodds",  
status=dt, counts=vm, groups=groups, anno=fit$genes)
```

2 Creating and sharing output

All interactive plots are automatically saved as html files in a “glimma-plots” folder that is created in the current working directory, unless specified otherwise using the `path` and `folder` arguments. By default MDS plots are saved as “MDS-Plot.html”, MD plots as “MD-Plot.html”, and XY plots “XY-Plot.html”. Alternate file names can be specified using the `html` argument. As each plot is created and saved, an html page is also launched automatically in your default web browser; `launch` can be set to `FALSE` if this is not desired.

Glimma’s interactive plots can be distributed to collaborators by sharing the complete “glimma-plots” folder with its contents. Note that sharing html files alone will not work. In an Rmarkdown analysis report the interactive plots can be included as links in their relevant sections

3 Multi-dimensional scaling plots

Interactive MDS plots show similarities between the transcriptional profile of samples via unsupervised clustering. Glimma's MDS plot can be created on expression data in the form of a numeric matrix, DGEList, Elist, or DESeqDataSet object. Raw counts in an DGEList are automatically converted by `glMDSPlot` into logCPM values using normalisation factors within the object. For an equivalent plot using an expression matrix, raw counts need to be manually converted to logCPM values. An example is shown below using the `cpm` function in edgeR which takes into account the normalisation factors stored within the DGEList.

```
lcpm <- cpm(rnaseq, log=TRUE, normalized.lib.sizes=TRUE)
glMDSPlot(lcpm, groups=groups)
```

The output contains two key components. On the left is an MDS plot showing two consecutive dimensions plotted against each other with each sample represented by a point in the plot. The distance between two samples reflect the *leading logFC* or typical logFC of genes separating the samples. By default the top 500 genes are used to calculate distances unless specified otherwise using the `top` argument. For more information on MDS plots, see `?limma::plotMDS`.

On the right, a barplot is displayed representing the proportion of variation in the data that is explained by the dimensions or eigenvectors in the MDS plot. Dimension 1 which explains the largest proportion of variation is associated with the first, left-most bar. The second bar is associated with dimensions 2, the third bar is for dimensions 3, and so on. Clicking on a bar on the page will highlight two consecutive bars and display the associated dimensions in the MDS plot.

Hovering your cursor over each of the points in the MDS plot brings up sample information such as sample labels and groups which can be specified using the `labels` and `groups` arguments. The coloring of points in the plot are associated with each unique group label. Typically `groups` would be a vector specifying the main condition by which samples are separated, but for more complex experimental designs a dataframe can also be used to represent multiple categorical variables.

To demonstrate this, a dataframe is created using genotype and sequencing lane information (all samples were sequenced on *lane 4* except for the last sample which was sequenced on *lane 3*). An interactive MDS plot is created by using the dataframe to define `groups`. The screen-captures in Figure 4 from the html output shows the switching of sample colors by genotype to sequencing lane, and a change in the displayed dimensions.

```
groups.df <- as.data.frame(cbind(
  genotype=as.character(groups),
  lane=c(rep(4,6),3)))
groups.df

##      genotype lane
## 1 Smchd1-null    4
## 2 Smchd1-null    4
## 3 Smchd1-null    4
## 4 Smchd1-null    4
```



```
## 5      WT      4
## 6      WT      4
## 7      WT      3

glMDSPlot(lcpm, groups=groups.df)
```

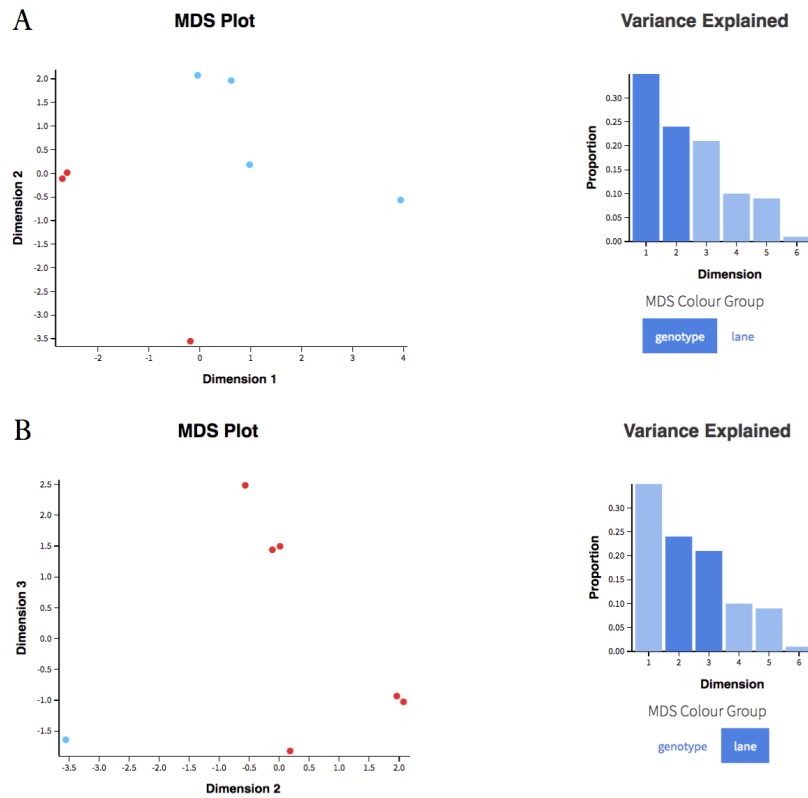


Figure 4: Interactive MDS plots showing (A) dimensions 1 and 2 with samples colored by group (or genotype) and (B) dimensions 2 and 3 with samples colored by sequencing lane.

4 Mean-difference plots

4.1 General

Mean-difference plots provide a visual summary of the results and are useful for highlighting genes with unusually large absolute logFCs amongst all of the genes that are tested. When “stand out” genes are spotted in the MD plot it is often of interest to see the expression of individual samples for that gene to check the consistency of expression within groups and for the potential of outliers. Glimma’s MD plot makes that connection between summarised results (across all genes) and individual sample expression (for any selected gene) so that the data can be interrogated more thoroughly by having the two plots side-by-side.

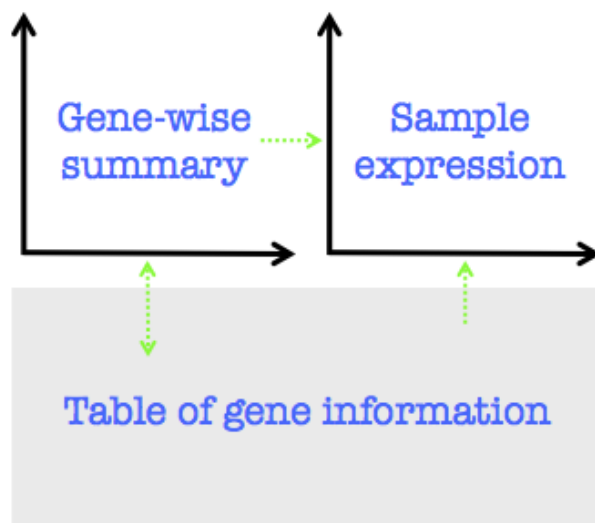


Figure 5: Layout of MD plots with three key components – two plots on top and a table below. Green arrows represent the direction of interaction between components.

The interactive MD plot contains three key components which interact with each other to show multiple aspects of the data in the one display. The layout of such a plot is shown in Figure 5. The main component is a plot of gene-wise summarised statistics which takes the top-left panel of the html page. Gene-wise logFCs are plotted against gene-wise average logCPM values where each point on the plot represents a single gene. Hovering your cursor over or clicking on a gene (or point) within the main plot brings the expression of each sample for the selected gene in a plot in the top-right panel. At the same time, associated gene information is displayed in the table below.

Users can simply scroll through the table looking for any gene that is of interest, or hone into specific genes or groups of genes using the search function in the table. Clicking on a gene (or row) in the table interacts with both of the plots simultaneously – the selected gene is highlighted in the MD plot and next to it, the expression each sample is displayed.

The order of genes displayed in the table can be re-ordered in an increasing or decreasing fashion by clicking on the header of a column. This is useful to see which genes have the smallest raw or adjusted p-value, or for sorting genes into those that are most up- or down-regulated in terms of logFC. The ordering function in the table used in conjunction with the search function can be especially powerful as an exploratory tool,

for example, one can search by a keyword of interest, say “structural maintenance” and then order the reduced table of genes by adjusted p-value.

When working with limma output, average expression values, logFCs and associated gene information are automatically extracted from MArrayLM objects. By default the last coefficient in the object is used unless specified otherwise using the `coef` argument. In it’s simplest form, `glMDSPlot` can take an MArrayLM object alone with `counts` unspecified, as shown in the R-code below. In this way, only the main plot and table will be displayed.

```
glMDSPlot(fit)
```

When it is used, `counts` can be raw or transformed counts (e.g. cpm or logCPM) that must have the same ordering of genes as in the main argument `x`. If raw counts are given, they can be transformed into logCPM values by setting `transform` to `TRUE`.

4.2 Plotting options

Sample expression can be sorted into groups using the `groups` argument, where `groups` is a vector matching in length and order to the samples (or columns) in `counts`. Typically `groups` will be a character or factor vector separating samples into different conditions, as demonstrated in Section 1. However, `groups` can also be a numeric vector associating expression values with a covariate of interest, for example, the age of mice at the time of RNA extraction.

```
groups.age <- runif(ncol(rnaseq), min=3, max=12)
groups.age

## [1] 3.6 10.9 11.2 8.9 7.6 10.4 9.5
```

In the main plot, up- and down-regulated genes can be highlighted using the `status` argument which is a vector containing integer values of -1 to represent down-regulated genes, 0 for no differential expression, and 1 for up-regulated genes. These values can be given in the form of a numeric vector that is of the same length and ordering of genes in `x`. Alternatively, if a matrix or a `TestResults` object is supplied, then the column specified by `coef` will be used to highlight genes. By default, down-regulated genes are colored in blue and up-regulated genes are in red. Alternatively, your own colors can be specified using the `cols` argument which accepts both R-defined colors such as “blue”, and numeric values which references your current color palette.

In the side sample expression plot, `side.main` specifies the column from table which is used as the main title, for example, `side.main= “GeneName”`. `side.xlab` and `side.ylab` is used to specify the labels for x- and y- axes. Sample labels which appear when clicking on or hovering over a point can be changed using the `samples` argument; and colors of points can be specified using the `sample.cols` argument. Other arguments include `jitter` which jitters points horizontally to minimise the amount of overlap (does not apply when `groups` is numeric), `side.log` which re-scales the y-axis to a log-scale (but does not transform the data), and `side.gridstep` which adds horizontal grid lines to the plot.

Using the age of mice in the sample expression plot, we demonstrate the use of some of the options described above (Figure 6). Notice that the color of points in both the MD plot and sample expression plot has changed, and more informative labels have been specified.

```
cols <- c("yellow", "blue", "magenta")
sample.cols <- c("limegreen", "purple")[groups]
glMDPlot(fit, status=dt, counts=vm, groups=groups.age,
  sample.cols=sample.cols, cols=cols,
  side.ylab="logCPM", side.xlab="Age (in months)",
  side.main="Symbols", main=colnames(dt))
```

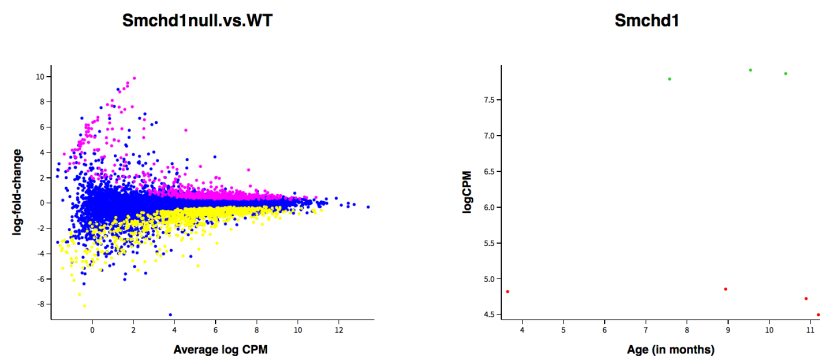


Figure 6: Interactive MD plot (left) where sample expression (right) has been stratified by age. The table is not displayed here to highlight changes to MD and sample expression plots.

4.3 Table options

Gene information is automatically extracted from `MArrayLM` and `DGEList` objects and displayed within the table, along with the values for average gene expression, logFC and adjusted *p*-value. `glMDPlot` does this by looking under the `$genes` slot of `x`.

Extra gene annotation can be added to the table using the `anno` argument. This would combine and display both the gene information from `x` and `anno`, where `anno` is a dataframe with the same ordering and number of genes as in `x`. To display specific columns in the table use the `display.columns` argument.

In the example below, we create extra gene annotation, where *ID* combines gene symbol with Entrez gene ID and *DE* specifies whether genes are downregulated, upregulated or not differentially expressed (notDE). Using `display.columns`, we display only *ID* and *DE*, and full gene names from `fit$genes`.

```
ID <- paste(fit$genes$Symbols, fit$genes$GeneID)
DE <- c("downregulated", "notDE", "upregulated")[as.factor(dt)]
anno <- as.data.frame(cbind(ID, DE))
head(anno)

##           ID           DE
## 1 Abca1 11303 downregulated
## 2 Abca2 11305          notDE
## 3 Abcb7 11306          notDE
## 4 Abcg1 11307 downregulated
```

```
## 5  Abi1 11308      notDE
## 6  Abl1 11350 downregulated

glMDPlot(fit, counts=vm, groups=groups, side.main="ID",
  anno=anno, display.columns=c("ID", "GeneName", "DE"))
```

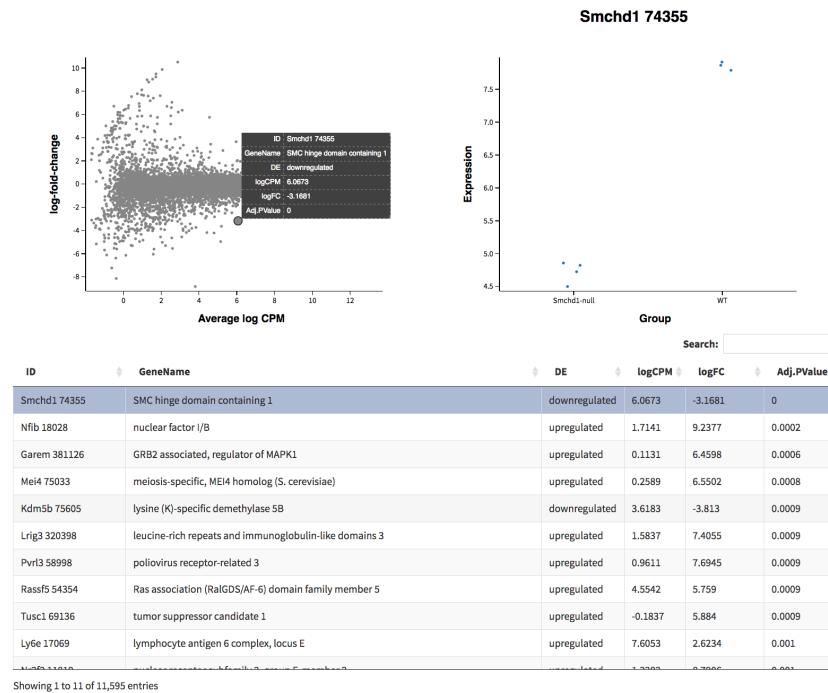


Figure 7: Interactive MD plot with changes to default gene information displayed in the table.

Adjusted p -values that are included in the table are automatically calculated using the Benjamini and Hochberg method [10] on raw p -values stored within `x`. Other multiple-testing correction methods that are available in `stats::p.adjust` can be specified to the `p.adj.method` argument.

When performing differential expression analyses using edgeR, the examples in this section would work by simply replacing limma's `MArrayLM` object with either of edgeR's `DGEEexact` or `DGELRT` object; the same goes for `DESeqDataSet` objects from a `DESeq2`-style analysis. `LogFC` values, average expression values and raw p -values are automatically extracted from all objects. Gene information, however, is only automatically extracted from the limma and edgeR objects but not for `DESeq2`. See Subsection ?? and ?? for examples using output from edgeR and `DESeq2`.

5 XY plots

Glimma's XY plots have the same layout as MD plots (Figure 5) but can be used to display any gene-wise summary statistic against any other gene-wise summary statistic as the main plot in the top left panel. The MD plot is essentially the XY plot with the x-component specified as average logCPM values and the y-component specified as logFC values. Since the XY plot is for general usage it works with basic R objects such as vectors, matrices and dataframes, rather than MArrayLM, DGEEexact, DGELRT or DESeqDataSet objects where gene information or raw p -values could otherwise be automatically extracted. The two main arguments in `glXYPlot` are `x` and `y`, both of which are numeric vectors of equal length. To create a volcano plot, we specify `x` as the logFC between Smchd1-null versus wildtype, and `y` as the log-odds that the gene is DE.

```
glXYPlot(x=fit$coef, y=fit$lod)
```

Since no other information is given to the function, genes are automatically assigned gene identifiers (GeneID) and labels remain as 'x' and 'y'. The labels can be specified as something more meaningful, such as 'logFC' and 'logodds' using the `xlab` and `ylab` arguments.

Other arguments in XY plot are analogous to those that are in the MD plot. In brief, `status` and `cols` are used to highlight DE genes in the main plot; `anno` adds gene information to the table where `display.columns` specifies the columns that are display; `counts` is used to add a plot of sample expression with `groups` separating observations into different conditions; `samples` and `sample.cols` labels and colors points in the sample expression plot, where `jitter` is applied to points avoid overlapping, and `side.main` is used as the title label.

Using some of the options mentioned, an enhanced version of the volcano plot is created using the R-code below (Figure 8).

```
glXYPlot(x=fit$coef, y=fit$lod, xlab="logFC", ylab="logodds",
  status=dt, anno=anno, side.main="ID",
  counts=vm, groups=groups, sample.cols=sample.cols)
```

The XY plot allows users to come up with an unlimited number of plotting combinations between any two gene-wise statistics for a dataset and relate these to sample-specific expression. It can also be used to compare results between datasets, for example the logFC from one experiment could be plotted against the logFC from a second experiment, with the corresponding sample expression presented in the left-hand panel as before.

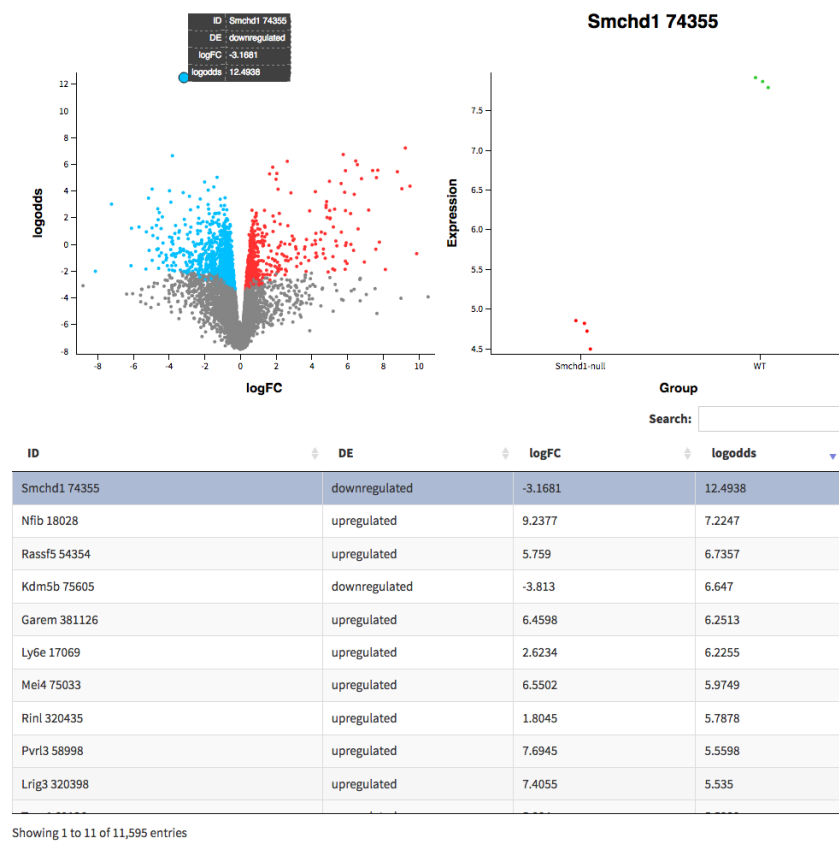


Figure 8: Interactive volcano plot (top left) with DE genes highlighted, and samples in the sample expression plot (top right) separated into genotype.

6 Using microarray data

Although Glimma was developed with RNA-sequencing data analyses in mind and designed to interact specifically with the limma, edgeR and DESeq2, it can be just as easily applied to microarray data especially when the data is processed with limma.

In this section, we demonstrate the usage of Glimma on an Illumina microarray dataset taken from a study on the *Ezh2* gene in mouse mammary epithelium [13]. The study includes two cell populations, one that is enriched for mammary stem cells (labeled as *DP*) and another that is enriched for luminal progenitor cells (labeled as *Lum*). In each population, there are three samples where the *Ezh2* gene has been knocked-out (labeled as *cre Ezh2*) and three wildtype samples (labeled as *ev*).

For this dataset, normexp [12] background correction and normalisation was carried out, and probes are removed from downstream analysis if they were not detected in any of the samples or are of low quality. Probes are considered as “detected” if they have a detection score of greater than 0.95, and are considered to have reasonable quality if it is graded as “Good” or better. The pre-processed expression data is available within Glimma as an EListRaw object and a targets file of associated sample information is included.

```
data(arraydata)
arrays <- arraydata$arrays
targets <- arraydata$targets
dim(arrays)

## [1] 10571    12

targets

##      Array SampleID      Condition      Chip Section Experiment
## 1      1      TB.05      DP ev 5233006042      A          2
## 2      2      TB.04 Lum cre Ezh2 5233006042      B          1
## 3      3      TB.06 DP cre Ezh2 5233006042      C          2
## 4      4      TB.01      DP ev 5233006042      D          1
## 5      5      TB.03      Lum ev 5233006042      E          1
## 6      6      TB.02 DP cre Ezh2 5233006042      F          1
## 7      7      TB.08 Lum cre Ezh2 5233006024      A          2
## 8      8      TB.7      Lum ev 5233006024      B          2
## 9      9      TB.11      Lum ev 5233006024      C          3
## 10     10     TB.12 Lum cre Ezh2 5233006024      D          3
## 11     11     TB.10 DP cre Ezh2 5233006024      E          3
## 12     12     TB.09      DP ev 5233006024      F          3
```

An MDS plot is created on the EList object with samples colored by sample condition, the Illumina beadchip on which samples were processed on, and experiment number. The plot shows that samples separate first by cell population (DP and Lum) over Dimension 1 (Figure 9A), and then separate by the beadchip and experiment over Dimension 2 (Figure 9B,C). Variations in the experimental design are easily explored using the interactive plot.


```
glMDSPlot(arrays, groups=targets[,c("Condition", "Chip", "Experiment")])
```

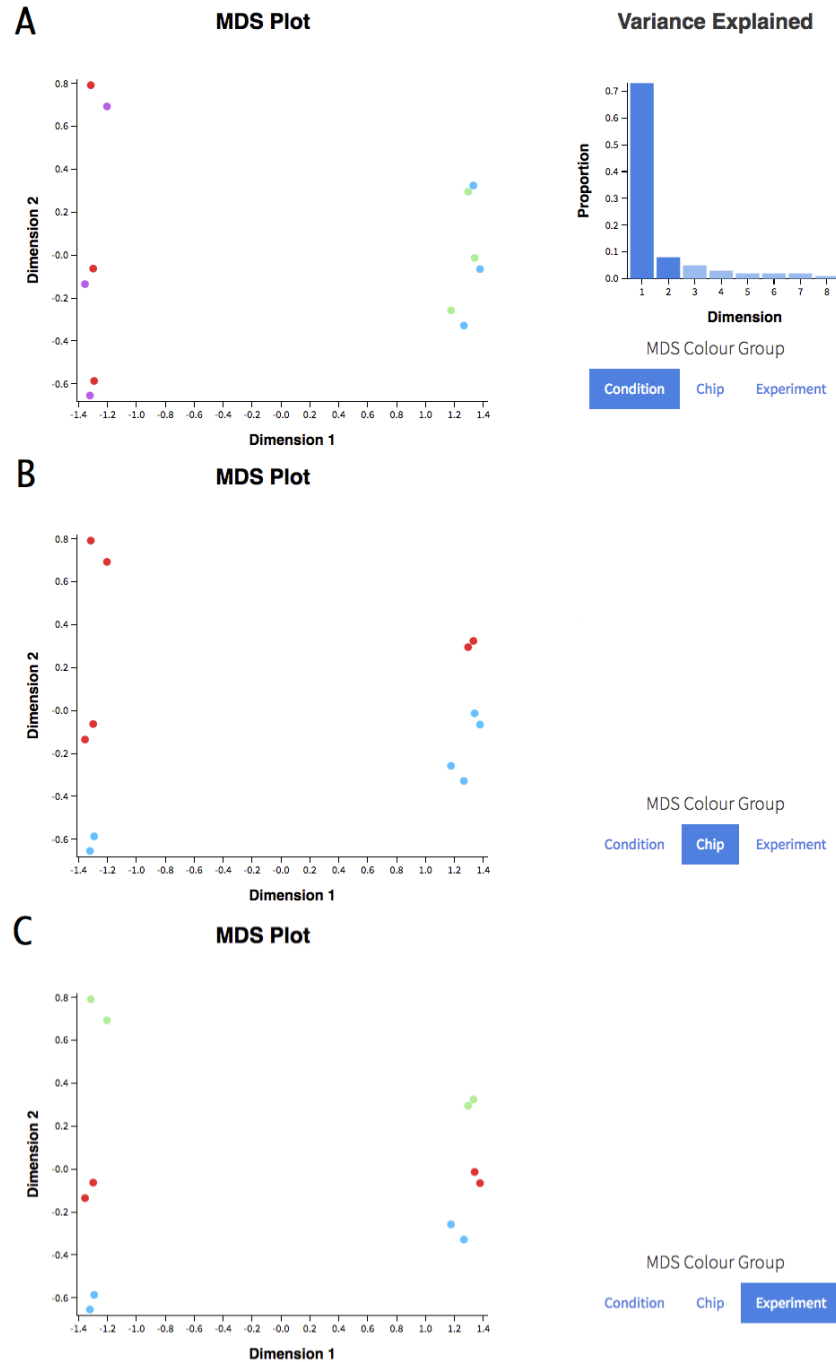


Figure 9: Interactive MDS plot with samples colored by condition (A), beadchip (B) and experiment (C). The plot showing the proportion of variation explained by each dimension (top right panel) has been hidden in panels B and C, to highlight the change in MDS Color Group.

Within each cell population we test for the probes that are DE for Ezh2 knock-out versus wildtype using a limma-style analysis. Sample conditions and experiment number is included as parameters used in linear modelling. Using an adjusted p -value of 0.1, 131

probes are detected as DE in the mammary stem cell-enriched population, and 85 probes are detected in the luminal population.

```
design <- model.matrix(~0+targets$Condition+as.factor(targets$Experiment))
contrasts <- cbind(
  DP_Ezh2KO.vs.WT=c(1,-1,0,0,0,0),
  Lum_Ezh2KO.vs.WT=c(0,0,1,-1,0,0))
fit <- lmFit(arrays, design)
fit <- contrasts.fit(fit, contrasts)
fit <- eBayes(fit)
dt <- decideTests(fit, p.value=0.1)
summary(dt)
```

	DP_Ezh2KO.vs.WT	Lum_Ezh2KO.vs.WT
## Down	37	65
## NotSig	10440	10486
## Up	94	20

A MD plot is created for each of the comparisons, with sample expression grouped by condition and colored by experiment number. Since gene identifiers are non-unique in microarray data, probe identifiers are used to label sample expression plots. Amongst the DE top genes that are displayed (as ranked by adjusted *p*-value), probe 6940037 is up-regulated in the comparison of Ezh2 knock-out versus wildtype in both cell populations (Figure ??).

```
sample.cols <- c("purple", "magenta", "green")[targets$Experiment]
for (COEF in 1:2) {
  glMDPlot(fit, status=dt, coef=COEF, main=colnames(fit)[COEF],
    counts=arrays, groups=targets$Condition, sample.cols=sample.cols,
    side.ylab="Log-expression", side.main="ProbeID")
}
```

To take a look at both comparisons at the same time, the logFC for DP Ezh2 knock-out versus wildtype is plotted against the logFC for Lum Ezh2 knock-out versus wildtype (Figure 11). Probes that are DE in either one of the comparisons are highlighted in the plot in black, and probes that are DE in both comparisons are highlighted in the plot in red. We search specifically for “Ltf” to find probe 6940037 using the table’s search bar. Probe 6940037 has the largest positive logFC in both comparisons. In general, logFCs in the two cell populations are positively correlated for the comparison between Ezh2 knock-out versus wildtype.

```
dt2 <- rep(0, nrow(dt))
dt2[rowSums(dt!=0)==1] <- -1
dt2[rowSums(dt!=0)==2] <- 1
table(dt2)
```

## dt2	-1	0	1
##	184	10371	16

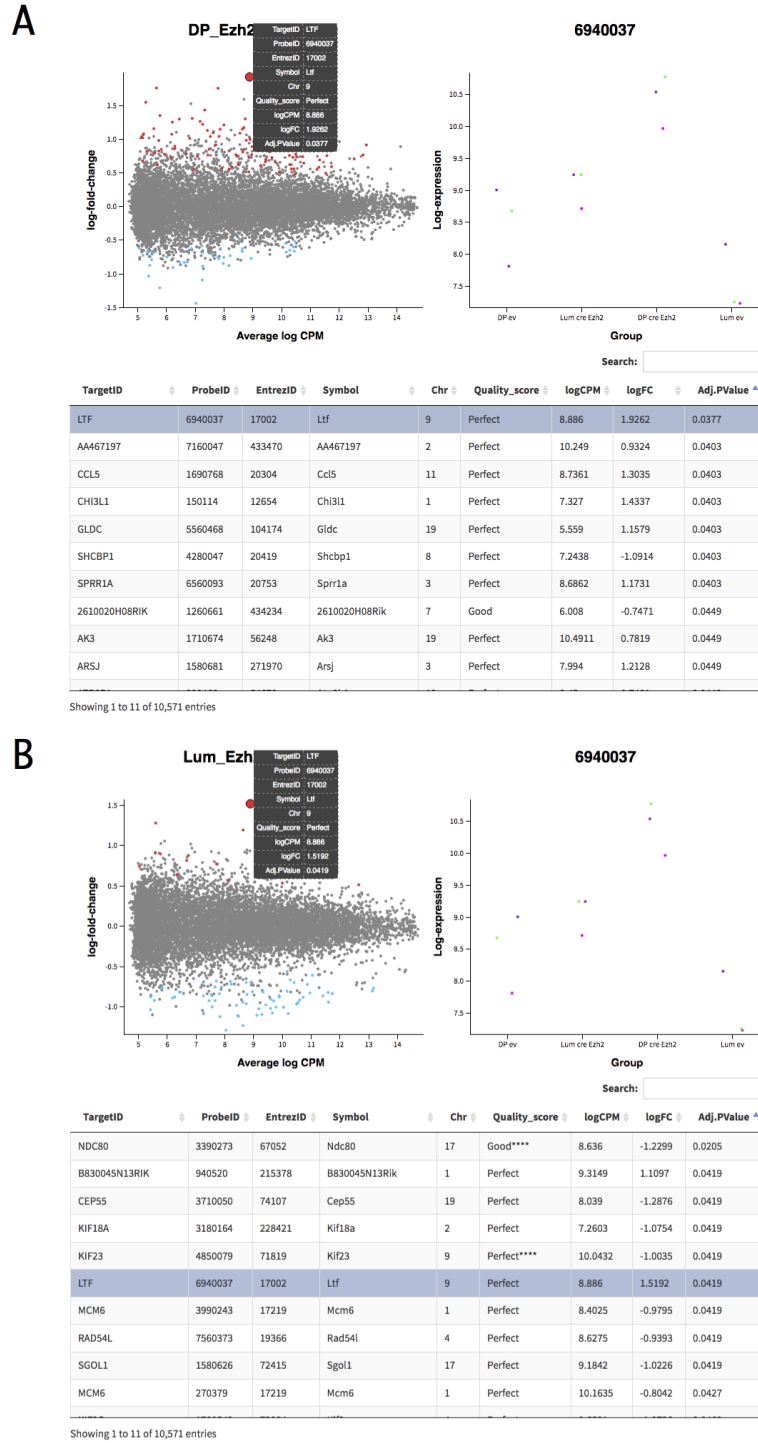


Figure 10: Interactive MD plot for the comparison between Ezh2 knock-out and wild-type for A) mammary stem cell-enriched samples and B) luminal populations, where up-regulated genes are colored in red and down-regulated genes are colored in blue. Samples in the sample expression plot are grouped by condition and colored by experiment number. Both tables in A) and B) show the top DE genes are ranked by adjusted p -value. Probe 6940037 for gene *Ltf* is amongst the top DE genes in both comparisons.

```
cols <- c("black", "grey", "red")
glXYPlot(fit$coef[,1], y=fit$coef[,2], xlab="DP", ylab="Lum",
  status=dt2, cols=cols, anno=fit$genes, side.main="ProbeID",
  counts=arrays, groups=targets$Condition, sample.cols=sample.cols,
  side.ylab="Log-expression", main="logFCs")
```

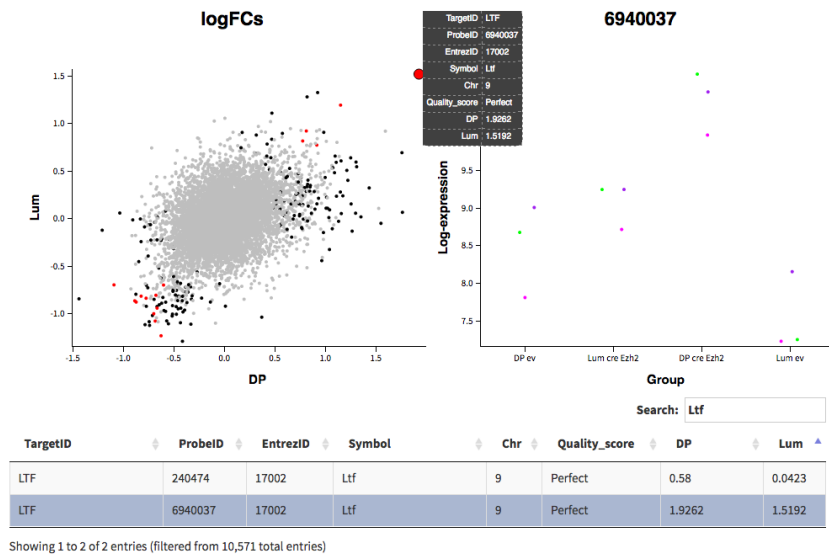


Figure 11: Interactive plot of logFCs for Ezh2 knock-out versus wildtype in DP (x-axis) and Lum (y-axis) in the top left panel. Probes that are DE in one comparison are highlighted in black, and probes that are DE in both comparisons are highlighted in red. Sample expression is separated into conditions and colored by experiment number. The table of results is restricted to those that match with “Ltf”.

7 Appendix

7.1 Extra mean-difference plots

7.1.1 edgeR-style analysis

In the R code below, DE analysis is carried out using edgeR's exact test method. A MD plot is created using a DGEEExact object, and *dt.edger* which is limma's TestResults object is used to highlight genes that are detected as differentially expressed. Since raw counts are given to the *glMDPlot* function, a logCPM transformation is carried out using the *transform* argument. For an analysis using edgeR's likelihood ratio tests, one can easily replace the DGEEExact object below with a DGEList object.

```
groups <- rnaseq$samples$group
design <- model.matrix(~groups)
colnames(design) <- c("WT", "Smchd1null.vs.WT")
rnaseq.edger <- estimateDisp(rnaseq, design=design)
fit.edger <- exactTest(rnaseq.edger)
dt.edger <- decideTestsDGE(fit.edger)
glMDPlot(fit.edger, status=dt.edger, counts=rnaseq, groups=groups, transform=TRUE)
```

7.1.2 DESeq2-style analysis

Differential expression analysis is carried out here using DESeq2. A MD plot is created using a DESeqResults object. Genes are highlighted (without distinction between up- or down-regulation) using the numeric vector *dt.deseq2*.

```
# BUG regarding the scale of sample expression
library(DESeq2)
rnaseq.deseq2 <- DESeqDataSetFromMatrix(
  rnaseq$counts, colData=rnaseq$samples, design=~group)
mcols(rnaseq.deseq2) <- DataFrame(mcols(rnaseq.deseq2), rnaseq$genes)
rnaseq.deseq2 <- DESeq(rnaseq.deseq2)
fit.deseq2 <- results(rnaseq.deseq2, contrast=c("group", "Smchd1-null", "WT"))
dt.deseq2 <- as.numeric(fit.deseq2$padj<0.05)
glMDPlot(fit.deseq2, status=dt.deseq2, counts=rnaseq, groups=groups, transform=FALSE,
  samples=colnames(rnaseq), anno=rnaseq$genes)
```

7.2 R session information

```
sessionInfo()

## R Under development (unstable) (2020-03-10 r77920)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server 2012 R2 x64 (build 9600)
##
```

```
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] edgeR_3.29.1 limma_3.43.5 Glimma_1.15.3 knitr_1.28
##
## loaded via a namespace (and not attached):
## [1] compiler_4.0.0 magrittr_1.5    tools_4.0.0    Rcpp_1.0.4     stringi_1.4.6
## [7] grid_4.0.0      locfit_1.5-9.4 jsonlite_1.6.1 stringr_1.4.0   xfun_0.12
## [13] evaluate_0.14
```

References

- [1] Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK. (2015) limma powers differential expression analyses for RNA-sequencing and microarray studies, *Nucleic Acids Research*, **43**(7):e47.
- [2] Powell DR. (2015) Degust: Visualize, explore and appreciate RNA-seq differential gene-expression data, <http://victorian-bioinformatics-consortium.github.io/degust/>.
- [3] Robinson MD, McCarthy DJ, Smyth GK. (2010) edgeR: a Bioconductor package for differential expression analysis of digital gene expression data, *Bioinformatics*, **26**(1):139–40.
- [4] Love MI, Huber W, Anders S. (2014) Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2, *Genome Biology*, **15**(12):550.
- [5] Liu R, Chen K, Jansz N, Blewitt ME, Ritchie, ME (2016) Transcriptional profiling of the epigenetic regulator Smchd1, *Genomics Data*, **7**:144–7.
- [6] Law CW, Chen Y, Shi W, Smyth GK (2014) Voom: precision weights unlock linear model analysis tools for RNA-seq read counts, *Genome Biology*, **15**:R29.
- [7] Liu R, Holik AZ, Su S, Jansz N, Chen K, Leong HS, Blewitt ME, Asselin-Labat ML, Smyth GK, Ritchie ME (2015) Why weight? Combining voom with estimates of sample quality improves power in RNA-seq analyses, *Nucleic Acids Research*, **43**(15):e97.
- [8] McCarthy DJ, Smyth GK (2009) Testing significance relative to a fold-change threshold is a TREAT, *Bioinformatics*, **25**(6):765–71.
- [9] Robinson MD, Oshlack A (2010) A scaling normalization method for differential expression analysis of RNA-seq data, *Genome Biology*, **11**:R25.
- [10] Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing, *Journal of the Royal Statistical Society Series B* **57**, 289–300.
- [11] Huber W, Carey V, Gentleman R, Anders S, Carlson M, Carvalho BS, Bravo HC, Davis S, Gatto L, Girke T, Gottardo R, Hahne F, Hansen KD, Irizarry RA, Lawrence M, Love MI, MacDonald J, Obenchain V, Oleś AK, Pagès H, Reyes A, Shannon P, Smyth GK, Tenenbaum D, Waldron L, Morgan M (2015) Orchestrating high-throughput genomic analysis with Bioconductor, *Nature Methods* **12**(2):151–121.
- [12] Shi W, Oshlack A, Smyth GK (2010) Optimizing the noise versus bias trade-off for Illumina Whole Genome Expression BeadChips, *Nucleic Acids Research* **38**:e204.
- [13] Pal B, Bouras T, Shi W, Vaillant F, Sheridan JM, Fu N, Breslin K, Jiang K, Ritchie ME, Young M, Lindeman GJ, Smyth GK, Visvader JE (2013) Global changes in the mammary epigenome are induced by hormonal cues and coordinated by Ezh2, *Cell Reports* **3**:411–426.