NOISeq: Differential Expression in RNA-seq

Sonia Tarazona (starazona@cipf.es)
Pedro Furió-Tarí (pfurio@cipf.es)
Alberto Ferrer (aferrer@eio.upv.es)
Ana Conesa (aconesa@cipf.es)

24 February 2014
(Version 2.6.0)

Contents

1 Introduction

This document intends to guide you through to the use of the R Bioconductor package NOISeq, for analyzing count data coming from next generation sequencing technologies. First, we describe the input data format. Then, we suggest you to explore the data using NOISeq functionalities in order to learn more about saturation, contamination or other biases in your data. Finally, we show how to compute differential expression between two experimental conditions. The differential expression method NOISeq and some of the plots included in the package were used in [1]. We also include in the package the new version of NOISeq for biological replicates, called NOISeqBIO.

NOISeq and NOISeqBIO differ from existing methods in that they are data-adaptive and nonparametric. Therefore, no distributional assumptions need to be done for the data and differential expression analysis may be carried on for both raw counts or previously normalized datasets.

We will use the "reduced" Marioni's dataset [2] as an example throughout this document. In Marioni's experiment, human kidney and liver RNA-seq samples were sequenced. There are 5 technical replicates per tissue. We selected chromosomes I to IV from the original data and removed genes with 0 counts in all samples and with no length information available. Note that this reduced dataset is only used to decrease the computing time while testing the examples. We strongly recommend to use the whole set of features (e.g. the whole genome) in real analysis.

The example dataset can be obtained by typing:

```
> library(NOISeq)
> data(Marioni)
```

2 Input data

NOISeq requires mainly two pieces of information that must be provided to the `readData` function: the expression data (`data`) and the factors defining the experimental groups to be studied or compared (`factors`). However, in order to normalize the count data or generate the exploratory plots included in the package for quality control, other additional annotations need to be provided such as the feature length, the GC content, the biological classification of the features (e.g. Ensembl biotypes), or the chromosome and position of each feature.

2.1 Expression data

The expression data must be provided in a matrix or a data.frame R object, having as many rows as the number of features to be studied and as many columns as the number of samples in the experiment. See in the following example part of the count data format from Marioni's dataset:

```
> head(mycounts)
```

	R1L1Kidney	R1L2Liver	R1L3Kidney	R1L4Liver	R1L6Liver	R1L7Kidney	R1L8Liver
ENSG00000177757	2	1	0	0	1	2	0
ENSG00000187634	49	27	43	34	23	41	35
ENSG00000188976	73	34	77	56	45	68	55
ENSG00000187961	15	8	15	13	11	13	12
ENSG00000187583	1	0	1	1	0	3	0
ENSG00000187642	4	0	5	0	2	12	1

	R2L2Kidney	R2L3Liver	R2L6Kidney
ENSG00000177757	1	1	3
ENSG00000187634	42	25	47
ENSG00000188976	70	42	82
ENSG00000187961	12	20	15
ENSG00000187583	0	2	3
ENSG00000187642	9	4	9

The expression data can be both read counts or normalized expression data such as RPKM values, and also any other normalized expression values.

2.2 Factors

Factors are given in a data.frame object as follows. For Marioni's data, we obviously have the factor "Tissue", but we can also define another toy factor ("TissueRun") just to show how the method works. The levels of the factor "Tissue" are "Kidney" and "Liver". The factor "TissueRun" combines the sequencing run (we have 2 sequencing runs) with the tissue and hence has four levels: "Kidney_1", "Liver_1", "Kidney_2" and "Liver_2".

Be careful here, the order of the elements of the factor must coincide with the order of the samples (columns) in the expression data file provided.

```
> myfactors = data.frame(Tissue = c("Kidney", "Liver", "Kidney", "Liver",
+     "Liver", "Kidney", "Liver", "Kidney", "Liver", "Kidney"), TissueRun = c("Kidney_1",
+     "Liver_1", "Kidney_1", "Liver_1", "Liver_1", "Kidney_1", "Liver_1",
+     "Kidney_2", "Liver_2", "Kidney_2"))
```

2.3 Additional annotation

To perform an exploratory analysis before or after the differential expression computation, some additional information is needed, as for example the feature length (which may be used also for normalization purposes), the GC content, the biological classification of the features (e.g. Ensembl biotypes), or the chromosome and position of the feature.

In the following lines you can see how the R objects containing such information should look like:

```
> head(mylength)

ENSG00000177757 ENSG00000187634 ENSG00000188976 ENSG00000187961 ENSG00000187583
           2464            4985            3870            4964            8507
ENSG00000187642
           6890

> head(mygc)

ENSG00000177757 ENSG00000187634 ENSG00000188976 ENSG00000187961 ENSG00000187583
           48.6            66.0            59.5            67.9            62.6
ENSG00000187642
           67.7

> head(mybiotypes)

 ENSG00000177757  ENSG00000187634  ENSG00000188976  ENSG00000187961  ENSG00000187583
       "lincRNA" "protein_coding" "protein_coding" "protein_coding" "protein_coding"
 ENSG00000187642
"protein_coding"

> head(mychroms)

                Chr GeneStart GeneEnd
ENSG00000177757   1    742614  745077
ENSG00000187634   1    850393  869824
ENSG00000188976   1    869459  884494
ENSG00000187961   1    885830  890958
ENSG00000187583   1    891740  900339
ENSG00000187642   1    900447  907336
```

Please note, that these objects might contain a different number of features and in different order than the expression data. However, it is important to specify the names or IDs of the features in each case so the package can properly match all this information. The length, GC content or biological groups (e.g. biotypes), could be vectors, matrices or data.frames. If they are vectors, the names of the vector must be the feature names or IDs. If they are matrices or data.frame objects, the feature names or IDs must be in the row names of the object. The same applies for chromosome information, which is also a matrix or data.frame.

2.4 Usage

For reading these data and to start working with all the functionalities from NOISeq package, the function `readData` has to be used. An example on how it works is shown below. The arguments `length`, `gc`, `biotype`, `chromosome` and `factors` receive the objects shown before:

```
> mydata <- readData(data = mycounts, length = mylength, gc = mygc, biotype = mybiotypes,
+     chromosome = mychroms, factors = myfactors)
> mydata
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 5088 features, 10 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: R1L1Kidney R1L2Liver ... R2L6Kidney (10 total)
  varLabels: Tissue TissueRun
  varMetadata: labelDescription
featureData
  featureNames: ENSG00000177757 ENSG00000187634 ... ENSG00000201145 (5088 total)
  fvarLabels: Length GC ... GeneEnd (6 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

The `readData` function returns an object of *Biobase's eSet* class. To see which information is included in this object, type for instance:

```
> str(mydata)
> head(assayData(mydata)$exprs)
> head(pData(mydata))
> head(featureData(mydata)@data)
```

Note that the features to be used by all the methods in the package will be those in the data expression object. If any of this features has not been included in the additional annotation (if given), the corresponding value will be NA.

It is possible to add information to an existing object. For instance, `noiseq` function accepts objects generated while using other packages such as `DESeq` package. In that case, annotations may not be included in the object. The `addData` function allows the user to add annotation data to the object. Imagine that you generated the data object like this:

```
> mydata <- readData(data = mycounts, chromosome = mychroms, factors = myfactors)
```

And now you want to include the length and the biotype of the features. Then you have to use the `addData` function:

```
> mydata <- addData(mydata, length = mylength, biotype = mybiotypes, gc = mygc)
```

IMPORTANT: Some packages such as *ShortRead* also use the `readData` function but with different input object and parameters. Therefore, some incompatibilities may occur that cause errors. To avoid this problem when loading simultaneously packages with functions with the same name but different use, the following command can be used: `NOISeq::readData` instead of simply `readData`.

3 Exploratory analysis

Data processing and experimental design in RNA-seq are not straightforward. From the biological samples to the expression quantification, there are many steps in which errors may be produced, despite of the many procedures developed to reduce noise at each one of these steps and to control the quality of the generated data. Therefore, once the expression levels (read counts) have been obtained, it is absolutely necessary to be able to detect potential biases or contamination before proceeding with further analysis such as differential expression. The

technology biases, such as the transcript length, GC content, PCR artifacts, uneven transcript read coverage, contamination by off-target transcripts or big differences in transcript distributions, are factors that interfere in the linear relationship between transcript abundance and the number of mapped reads at a gene locus (counts).

In this section, we present a set of plots to explore the count data that may be helpful to detect these potential biases so an appropriate normalization procedure can be chosen. For instance, these plots will be useful for seeing which kind of features (e.g. genes) are being detected in our RNA-seq samples and with how many counts, which technical biases are present, etc.

As it will be seen at the end of this section, it is also possible to generate a report in a PDF file including all these exploratory plots for the comparison of two samples or two experimental conditions.

3.1 Generating data for exploratory plots

There are several types of exploratory plots that can be obtained. They will be described in detail in the following sections. To generate any of these plots, first of all, `dat` function must be applied on the input data to obtain the information to be plotted. The user must specify the type of plot the data are to be computed for (argument `type`). Once the data for the plot have been generated with `dat` function, the plot will be drawn with the *explo.plot* function. For instance:

```
> myexplodata <- dat(mydata, type = "biodetection")

[1] "Biotypes detection is to be computed for:"
 [1] "R1L1Kidney" "R1L2Liver"  "R1L3Kidney" "R1L4Liver"  "R1L6Liver"  "R1L7Kidney" "R1L8Liver"
 [8] "R2L2Kidney" "R2L3Liver"  "R2L6Kidney"

> explo.plot(myexplodata)
```

To save the data in a user-friendly format, see the following example on the `dat2save` function:

```
> mynicedata <- dat2save(myexplodata)
```

We have grouped the exploratory plots in three categories according to the different questions that may arise during the quality control of the expression data:

- Biotype detection: Which kind of features are being detected? Is there any abnormal contamination in the data?
- Sequencing depth & Expression Quantification: Would it be better to increase the sequencing depth to detect more features? Are there too many features with low counts? Are the samples very different regarding the expression quantification?
- Sequencing bias detection: Should the expression values be corrected for the length or GC content bias? Should a normalization procedure be applied to account for the differences among RNA composition among samples?

3.2 Biotype detection

If a biological classification of the features is provided (e.g. Ensembl biotypes), the following plots are to see which kind of features are being detected. In general, when features are genes, it is expected that most of them will be protein-coding.

Fig. 1 shows the "biodetection" plot. The bar diagram shows the percentage of each biotype in the genome (i.e. in the whole set of features provided), which proportion has been detected in our sample (with number of counts higher than `k`=0) and the percentage of each biotype within the sample. The vertical green line separates the most abundant biotypes (in the left-hand side, corresponding to the left axis scale) from the rest (in the right-hand side, corresponding to the right axis scale).

This is an example on how to use the corresponding functions prior to generate the data to be plotted and then to draw the plot. Since factor=NULL, the data for the plot are computed separately for each sample. If the factor is a character vector indicating the experimental condition for each sample, the samples are aggregated within each condition by summing up the counts, and the data for the plot are computed per condition. In this case, samples in columns 1 and 2 from expression data are plotted and the features (genes) are considered to be detected if having a number of counts higher than k=0:

```
> mybiodetection <- dat(mydata, k = 0, type = "biodetection", factor = NULL)
```

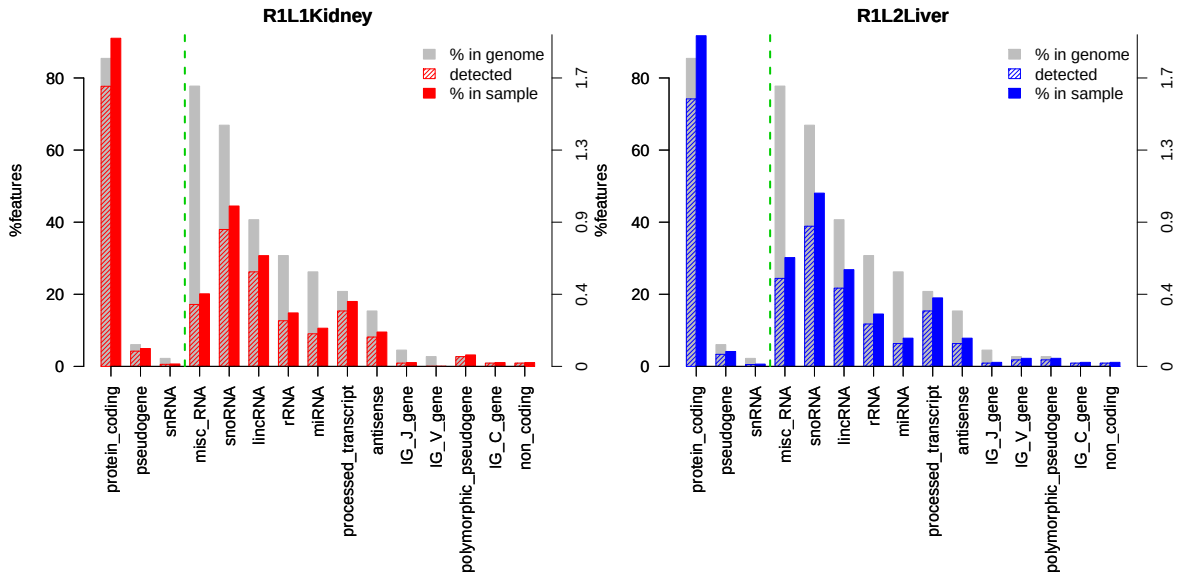

Figure 1: Biodetection plot

```
[1] "Biotypes detection is to be computed for:"
 [1] "R1L1Kidney" "R1L2Liver"  "R1L3Kidney" "R1L4Liver"  "R1L6Liver"  "R1L7Kidney" "R1L8Liver"
 [8] "R2L2Kidney" "R2L3Liver"  "R2L6Kidney"
```

```
> par(mfrow = c(1, 2))
> explo.plot(mybiodetection, samples = c(1, 2))
```

The "countsbio" plot, when providing the biological classification for the features, allows to see how the counts are distributed within each biological group, as it is shown in Fig. 2 and also (in the upper side of the plot) the number of detected features. The following code was used to draw the figure. Again, data are computed per sample because no factor was specified (factor = NULL). With *explo.plot* and "countsbio" data, two different plots may be generated. In this case, we chose the "boxplot" type, that shows the distribution of counts per biotype for samples 1 and 2 and for all the biotypes (because `toplot` parameter is 1).

```
> mycountsbio = dat(mydata, factor = NULL, type = "countsbio")
```

```
[1] "Count distributions are to be computed for:"
 [1] "R1L1Kidney" "R1L2Liver"  "R1L3Kidney" "R1L4Liver"  "R1L6Liver"  "R1L7Kidney" "R1L8Liver"
 [8] "R2L2Kidney" "R2L3Liver"  "R2L6Kidney"
```

```
> explo.plot(mycountsbio, toplot = 1, samples = 1, plottype = "boxplot")
```

3.3 Sequencing depth & Expression Quantification

The plots in this section can be generated by only providing the expression data, since no other biological information is required. Their purpose is to assess if the sequencing depth of the samples is enough to detect the features of interest and to get a good quantification of their expression.

"Saturation" plots allow you to know how many of the features in the genome are being detected with more than a given number of counts (e.g. k=0 or k=5) and also see how this number of detected features would change when increasing sequencing depth. This plot can be generated by considering either all the features or for each biological group (biotypes), if available. First, we have to generate the saturation data with the function `dat` and then we can use the resulting object to obtain, for instance, the plots in Fig. 3 and 4 by applying `explo.plot` function. When the number of samples to plot is 1 or 2, a right axis and bars are drawn to show the number of new features detected when increasing the sequencing depth in one million of reads. If more than 2 samples are to be plotted, it is difficult to visualize the newdetection bars, so the plot will only show the lines indicating the number of detected features at each sequencing depth.

```
> mysaturation = dat(mydata, k = 0, ndepth = 7, type = "saturation")
> explo.plot(mysaturation, toplot = 1, samples = 1:2, yleftlim = NULL, yrightlim = NULL)
```

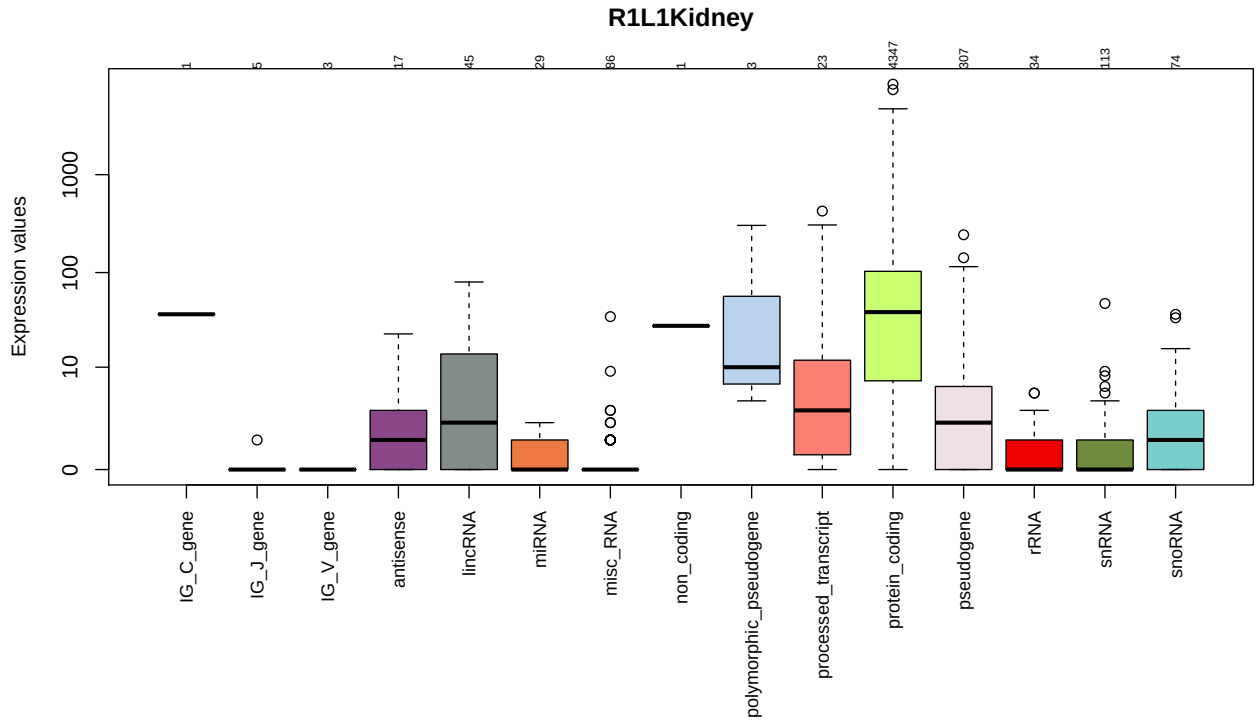

Figure 2: Count distribution per biotype in two samples (for genes with more than 0 counts). At the upper part of the plot, the number of detected features within each biotype group is displayed.

```
> explo.plot(mysaturation, toplot = "protein_coding", samples = 1:4)
```

The plot in Fig. 3 has been computed for all the features (without specifying a biotype) and for two of the samples. Left Y axis shows the number of detected genes with more than 0 counts at each sequencing depth, represented by the lines. The solid point in each line corresponds to the real available sequencing depth. The other sequencing depths are simulated from this total sequencing depth. The bars are associated to the right Y axis and show the number of new features detected per million of new sequenced reads at each sequencing depth. The legend in the gray box also indicates the percentage of total features detected with more than $k = 0$ counts at the real sequencing depth.

It is possible to compare more than 2 samples, but then the information in the right axis is not provided as it would be difficult to distinguish so many bars. You can compare up to twelve samples. In Fig. 4, four samples are compared and we can see, for instance, that in kidney samples the number of detected features is higher than in liver samples.

It is also interesting to visualize the count distribution for all the samples, either for all the features or for the features belonging to a certain biological group. Fig. 5 shows this information for the biotype "protein_coding", which can be generated with the following code on the "countsbio" object obtained in the previous section.

```
> explo.plot(mycountsbio, toplot = "protein_coding", samples = NULL, plottype = "boxplot")
```

Features with low counts are, in general, less reliable and may introduce noise in the data that makes more difficult to extract the relevant information, for instance, the differentially expressed features. In fact, we include some methods in NOISeq package to filter out these low count features and the sensitivity plot in Fig. 6 indicates the proportion of such features that are present in our data and also, if necessary, may help to decide about the filtering method to be applied. In this sensitivity plot, the bars show the percentage of features within each sample having more than 0 counts per million (CPM), or more than 1, 2, 5 and 10 CPM. The horizontal lines are the corresponding percentage of features with those CPM in at least one of the samples. In the upper side of the plot, the sequencing depth of each sample (in million reads) is given. The following code is for drawing this figure.

```
> explo.plot(mycountsbio, toplot = 1, samples = NULL, plottype = "barplot")
```

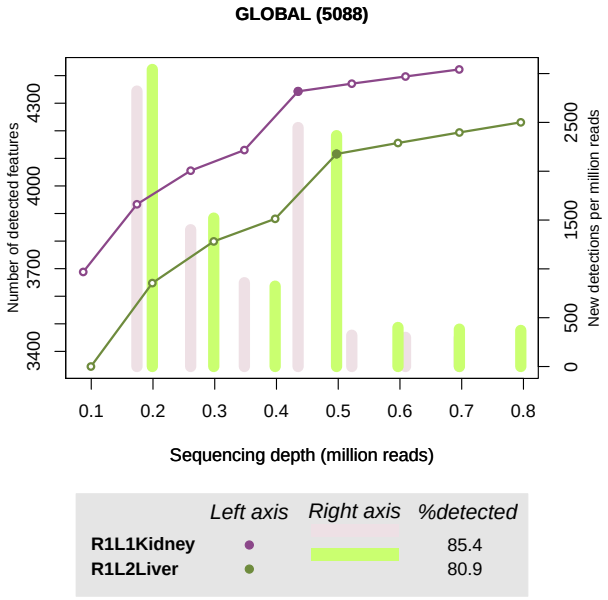

Figure 3: Global saturation plot to compare two samples of kidney and liver, respectively.

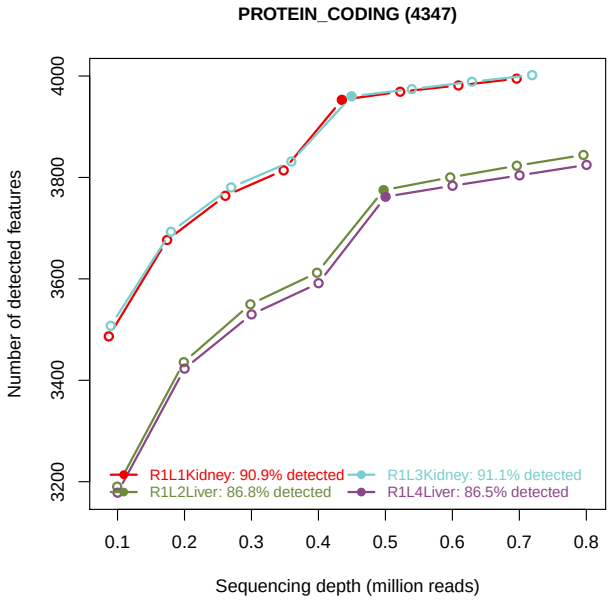

Figure 4: Saturation plot for protein-coding genes to compare 4 samples: 2 of kidney and 2 of liver.

3.4 Sequencing bias detection

Prior to perform further analyses such as differential expression, it is essential to normalize data to make the samples comparable and remove the effect of technical biases from the expression estimation. The plots presented in this section are very useful for detecting the possible biases in the data. In particular, the biases that can be studied are: the feature length effect, the GC content and the differences in RNA composition. In addition, these are diagnostic plots, which means that they are not only descriptive but an statistical test is also conducted to help the user to decide whether the bias is present and data needs normalization or not.

3.4.1 Length bias

The "lengthbias" plot describes the relationship between the feature length and the expression values. Hence, the feature length must be included in the input object created using the `readData` function. The data for this plot

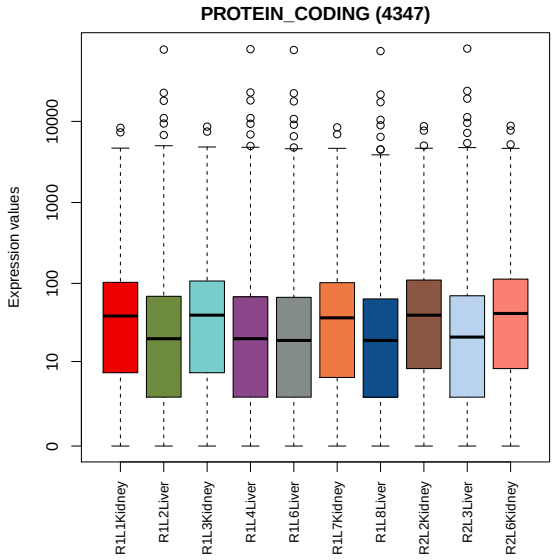

Figure 5: Distribution of counts for protein coding genes in all samples.

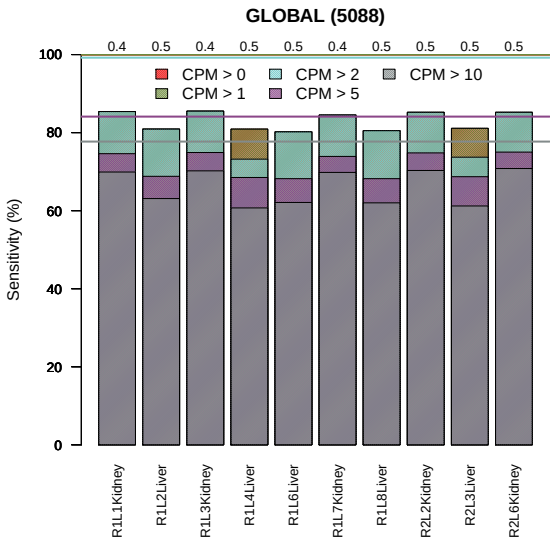

Figure 6: Number of features with low counts for each sample.

is generated as follows. The length is divided in intervals (bins) containing 200 features and the middle point of each bin is depicted in X axis. For each bin, the 5% trimmed mean of the corresponding expression values is computed and depicted in Y axis. If the number of samples or conditions to appear in the plot is 2 or less and no biotype is specified (toplot = "global"), a diagnostic test is provided. A cubic spline regression model is fitted to explain the relationship between length and expression. Both the model p-value and the coefficient of determination (R2) are shown in the plot as well as the fitted regression curve. If the model p-value is significant and R2 value is high (more than 70%), the expression depends on the feature length and the curve shows the type of dependence.

Fig. 7 shows an example of this plot. In this case, the "lengthbias" data were generated for each condition (kidney and liver) using the argument `factor`.

```
> mylengthbias = dat(mydata, factor = "Tissue", type = "lengthbias")
> explo.plot(mylengthbias, samples = NULL, toplot = "global")
```

More details about the fitted spline regression models can be obtained by using the `show` function as per below:

```
> show(mylengthbias)
```

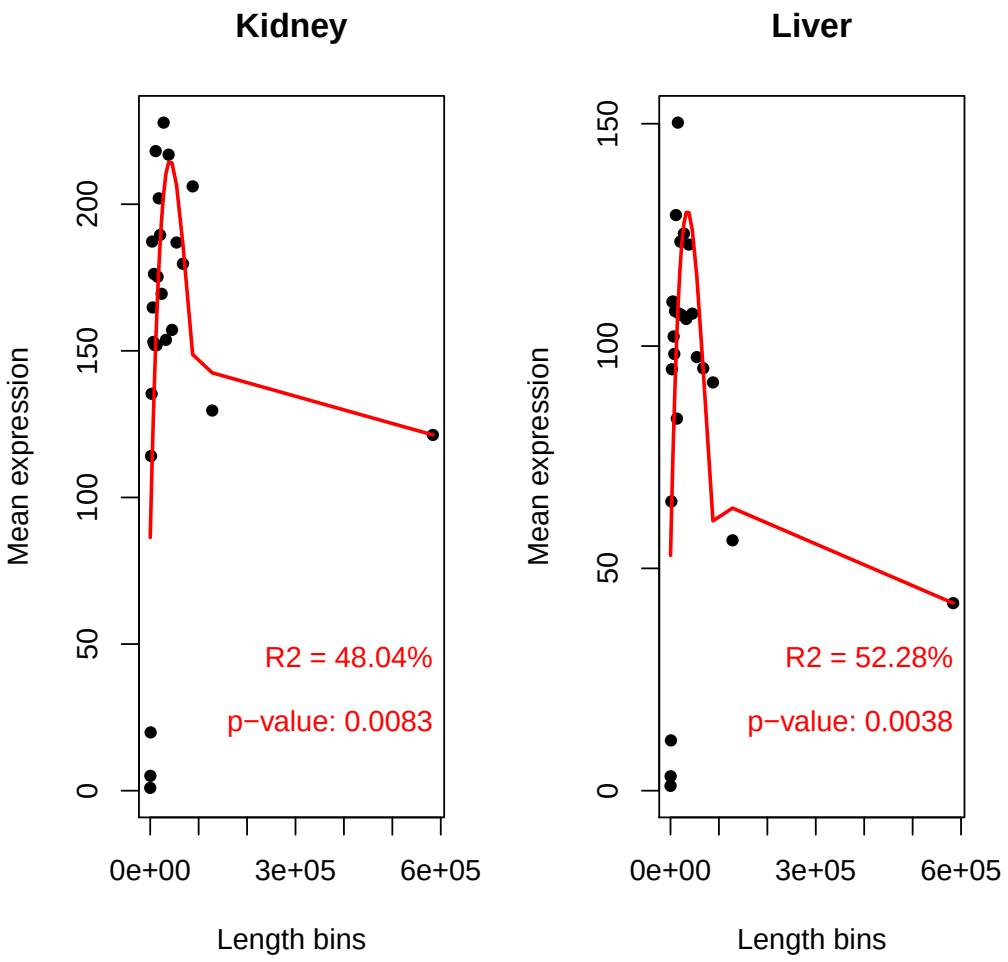

Figure 7: Gene length versus expression.

```
[1] "Kidney"

Call:
lm(formula = datos[, i] ~ bx)

Residuals:
   Min     1Q Median     3Q    Max
-85.40 -19.60   3.05  25.85  74.83

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    121.3       48.6    2.49   0.0215 *
bx1            -35.0       52.2   -0.67   0.5108
bx2            269.2       88.4    3.05   0.0064 **
bx3          -1301.1      719.7   -1.81   0.0857 .
bx4           6292.4     4655.4    1.35   0.1916
bx5               NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 48.6 on 20 degrees of freedom
Multiple R-squared:  0.48,        Adjusted R-squared:  0.376
F-statistic: 4.62 on 4 and 20 DF,  p-value: 0.00833
```

```
[1] "Liver"

Call:
lm(formula = datos[, i] ~ bx)

Residuals:
   Min     1Q Median     3Q    Max
 -51.8  -18.2    0.0   21.3   42.3

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)     42.2       29.7    1.42  0.17073
bx1             10.7       31.9    0.34  0.73952
bx2            222.1       54.0    4.12  0.00054 ***
bx3          -1141.7      439.3   -2.60  0.01716 *
bx4           5758.1     2841.2    2.03  0.05624 .
bx5               NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.7 on 20 degrees of freedom
Multiple R-squared:  0.523,        Adjusted R-squared:  0.427
F-statistic: 5.48 on 4 and 20 DF,  p-value: 0.00382
```

3.4.2 GC content bias

The "GCbias" plot describes the relationship between the feature GC content and the expression values. Hence, the feature GC content must be included in the input object created using the `readData` function. The data for this plot is generated in an analogous way to the "lengthbias" data. The GC content is divided in intervals (bins) containing 200 features. The middle point of each bin is depicted in X axis. For each bin, the 5% trimmed mean of the corresponding expression values is computed and depicted in Y axis. If the number of samples or conditions to appear in the plot is 2 or less and no biotype is specified (toplot = "global"), a diagnostic test is provided. A cubic spline regression model is fitted to explain the relationship between GC content and expression. Both the model p-value and the coefficient of determination (R2) are shown in the plot as well as the fitted regression curve. If the model p-value is significant and R2 value is high (more than 70%), the expression will depend on the feature GC content and the curve will show the type of dependence.

An example of this plot is in Fig. 8. In this case, the "GCbias" data were generated for each condition (kidney and liver) using the argument `factor`.

```
> myGCbias = dat(mydata, factor = "Tissue", type = "GCbias")
> explo.plot(myGCbias, samples = NULL, toplot = "global")
```

3.4.3 RNA composition

When two samples have different RNA composition, the distribution of sequencing reads across the features is different in such a way that although a feature had the same number of read counts in both samples, it would not mean that it was equally expressed in both. To check if this bias is present in the data, the "cd" plot and the correponding diagnostic test can be used. In this case, each sample s is compared to the reference sample r (which can be arbitrarily chosen). To do that, M values are computed as $log2(counts_s = counts_r)$. If no bias is present, it should be expected that the median of M values for each comparison is 0. Otherwise, it would be indicating that expression levels in one of the samples tend to be higher than in the other, and this could lead to false discoveries when computing differencial expression. Confidence intervals for the M median are also computed by bootstrapping. If value 0 does not fall inside the interval, it means that the deviation of the sample with regard to the reference sample is statistically significant. Therefore, a normalization procedure such as Upper Quartile, TMM or DESeq should be used to correct this effect and make the samples comparable before computing differential expression. Confidence intervals can be visualized by using `show` function.

See below an usage example and the resulting plot in Fig. 9. It must be indicated if the data provided are already normalized (norm = TRUE) or not (norm = FALSE). The reference sample may be indicated with the

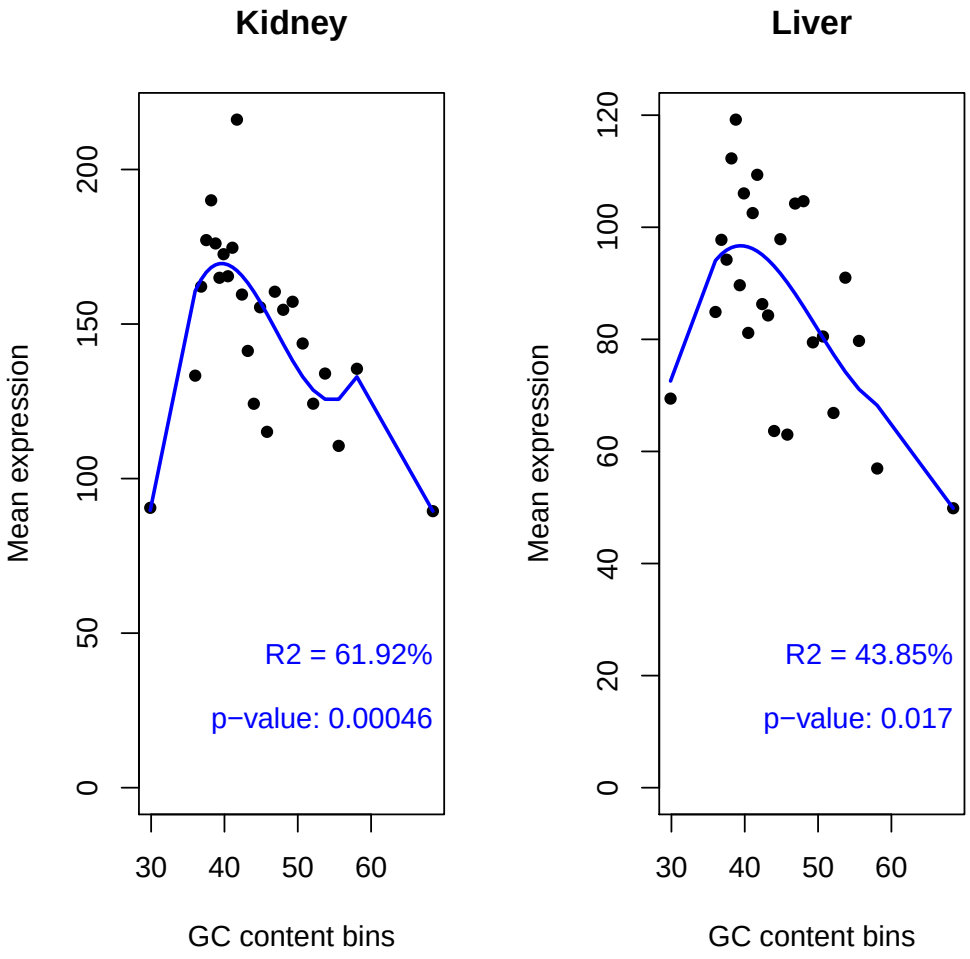

Figure 8: Gene GC content versus expression.

refColumn parameter (by default, the first column is used). Additional plot parameters may also be used to modify some aspects of the plot.

```
> mycd = dat(mydata, type = "cd", norm = FALSE, refColumn = 1)

[1] "Reference sample is: R1L1Kidney"
[1] "Confidence intervals for median of M:"
           0.28%                  99.7%                 Diagnostic Test
R1L2Liver  "-0.892840806813954"  "-0.758762774189429"  "FAILED"
R1L3Kidney "-0.0471703644087826" "-0.0471703644087824" "FAILED"
R1L4Liver  "-0.879624287764932"  "-0.752720648283132"  "FAILED"
R1L6Liver  "-0.908706394226066"  "-0.764019446494524"  "FAILED"
R1L7Kidney "0.0348451027997053"  "0.0348451027997056"  "FAILED"
R1L8Liver  "-0.898961552063353"  "-0.758871207596215"  "FAILED"
R2L2Kidney "-0.0850229820491386" "-0.0458980863022721" "FAILED"
R2L3Liver  "-0.876596368757286"  "-0.74179516290269"   "FAILED"
R2L6Kidney "-0.070582161929515"  "-0.035089941680745"  "FAILED"
[1] "Diagnostic test: FAILED. Normalization is required to correct this bias."

> explo.plot(mycd)
```

In the plot can be seen that the M median is deviated from 0 in most of the cases. This is corraborated by the confidence intervals for the M median.

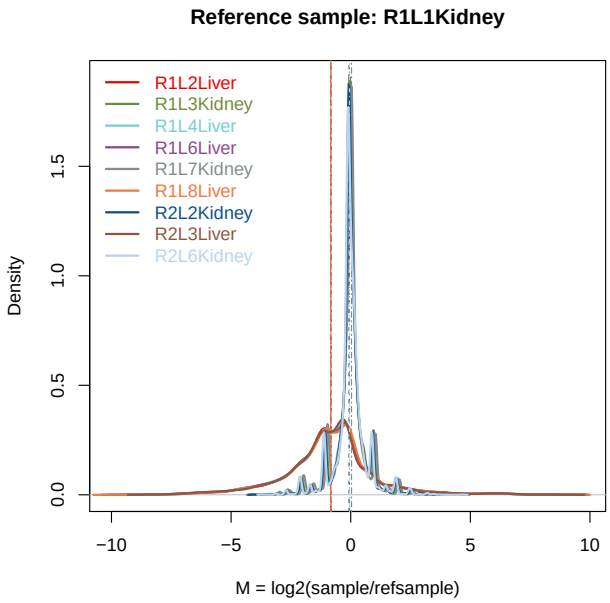

Figure 9: RNA composition plot

3.5 Quality Control report

The `QCreport` function allows the user to quickly generate a pdf report showing the exploratory plots described in this section to compare either two samples (if `factor` is NULL) or two experimental conditions (if `factor` is indicated). Depending on the biological information provided (biotypes, length or GC content), the number of plots included in the report may differ.

```
> QCreport(mydata, samples = NULL, factor = "Tissue")
```

4 Differential expression

The NOISeq package computes differential expression between two experimental conditions given the expression level of the considered features. The package includes three non-parametric approaches for differential expression analysis: NOISeq-real for technical replicates, NOISeq-sim for experiments without replications and NOISeqBIO, which is optimized for the use of biological replicates. All three methods take read counts from RNA-seq as the expression values, in addition to previously normalized data and read counts from other NGS technologies.

The normalization step is very important in order to make the samples comparable and to remove possibles biases in the data. It might also be useful to filter out low expression data prior to differential expression analysis, since they are less reliable and may introduce noise in the analysis.

Next sections explain how to use NOISeq package to normalize and filter data, to apply the three differential expression analysis and to visualize the results.

4.1 Normalization

We strongly recommend to normalize the counts to correct, at least, sequencing depth bias. The normalization techniques implemented in `NOISeq` are RPKM [3], Upper Quartile [4] and TMM, which stands for Trimmed Mean of M values [5].

Normalization functions `rpkm`, `tmm` and `uqua` can be used by themselves. Please, find below some examples on how to apply them:

```
> myRPKM = rpkm(assayData(mydata)$exprs, long = mylength, k = 0, lc = 1)
> myUQUA = uqua(assayData(mydata)$exprs, long = mylength, lc = 0.5, k = 0)
> myTMM = tmm(assayData(mydata)$exprs, long = 1000, lc = 0)
> head(myRPKM[, 1:4])
```

```
                R1L1Kidney R1L2Liver R1L3Kidney R1L4Liver
ENSG00000177757       1.87     0.816      0.000     0.000
ENSG00000187634      22.60    10.891     19.193    13.636
ENSG00000188976      43.36    17.664     44.265    28.926
ENSG00000187961       6.95     3.241      6.724     5.236
ENSG00000187583       0.27     0.000      0.262     0.235
ENSG00000187642       1.33     0.000      1.615     0.000
```

If the length of the features is provided and any of the three normalization methods is chosen, then the expression values are divided by $(length/1000)^{lc}$ ($lc = 1$ for RPKM method). Thus, although Upper Quartile and TMM methods themselves do not correct for the length of the features, NOISeq allows the users to combine these normalization procedures with an additional length correction whenever the length information is available. If $lc = 0$, no length correction is applied.

In addition, when using NOISeq or NOISeqBIO to compute differential expression, normalization can be done automatically by choosing the corresponding value for the parameter "norm". These methods also accept expression values normalized with other packages or procedures. Please, read sections 4.3 and 4.3.3 to get more information on this issue.

4.2 Low count filter

Excluding features with low counts improves, in general, differential expression results, no matter the method being used, since noise in the data is reduced. However, the best procedure to filter these low count features has not been yet decided nor implemented in the differential expression packages. NOISeq includes three methods to filter out features with low counts:

- **CPM** (method 1): The user chooses a value for the parameter counts per million (CPM) in a sample under which a feature is considered to have low counts. The cutoff for a condition with s samples is $CPM \times s$. Features with sum of expression values below the condition cutoff in all conditions are removed. Also a cutoff for the coefficient of variation (in percentage) per condition may be established to eliminate features with inconsistent expression values.
- **Wilcoxon test** (method 2): For each feature and condition, $H_0 : m = 0$ is tested versus $H_1 : m > 0$, where m is the median of counts per condition. Features with p-value > 0.05 in all conditions are filtered out. This method is only recommended when the number of replicates per condition is at least 5.
- **Proportion test** (method 3): Similar procedure to the Wilcoxon test but testing $H_0 : p = p_0$ versus $H_1 : p > p_0$, where p is the feature relative expression and $p_0 = CPM/10^6$. Features with p-value > 0.05 in all conditions are filtered out.

This is an usage example of function `filtered.data` directly on count data with CPM method (method 1):

```
> myfilt = filtered.data(mycounts, factor = myfactors$Tissue, norm = FALSE,
+     depth = NULL, method = 1, cv.cutoff = 100, cpm = 1)

[1] "Filtering out low count features..."
[1] "4406 features are to be kept for differential expression analysis with filtering method 1"
```

By default, NOISeqBIO includes the possibility of filtering data. The method to be used is indicated with the "filter" parameter.

4.3 NOISeq

NOISeq method was designed to compute differential expression on data with technical replicates (NOISeq-real) or no replicates at all (NOISeq-sim). If there are technical replicates available, it summarizes them by summing up them. It is also possible to apply this method on biological replicates, that are averaged instead of summed. However, for biological replicates we strongly recommend NOISeqBIO. NOISeq computes the following differential expression statistics for each feature: M (which is the log_2-ratio of the two conditions) and D (the value of the difference between conditions). Expression levels equal to 0 are replaced with the given constant $k > 0$, in order to avoid infinite or undetermined M-values. A feature is considered to be differentially expressed if its corresponding M and D values are likely to be higher than in noise. Noise distribution is obtained by comparing all pairs of replicates within the same condition. The corresponding M and D values are pooled together to generate the distribution. Changes in expression between conditions with the same magnitude than changes in expression between replicates within the same condition should not be considered as differential expression.

Thus, by comparing the (M, D) values of a given feature against the noise distribution, NOISeq obtains the "probability of differential expression" for this feature. If the odds Pr(differential expression)/Pr(non-differential expression) are higher than a given threshold, the feature is considered to be differentially expressed between conditions. For instance, an odds value of 4:1 is equivalent to q = Pr(differential expression) = 0.8 and it means that the feature is 4 times more likely to be differentially expressed than non-differentially expressed. The NOISeq algorithm compares replicates within the same condition to estimate noise distribution (NOISeq-real). When no replicates are available, NOISeq-sim simulates technical replicates in order to estimate the differential expression probability. Please remember that to obtain a really reliable statistical results, you need biological replicates. NOISeq-sim simulates technical replicates from a multinomial distribution, so be careful with the interpretation of the results when having no replicates, since they are only an approximation and are only showing which genes are presenting a higher change between conditions in your particular samples.

Table 1 summarizes all the input options and includes some recommendations for the values of the parameters when using NOISeq:

Table 1: Possibilities for the values of the parameters

Method	**Replicates**	**Counts**	**norm**	**k**	**nss**	**pnr**	**v**
NOISeq-real	Technical/Biological	Raw Normalized	rpkm, uqua, tmm n	0.5 NULL	0	-	-
NOISeq-sim	None	Raw Normalized	rpkm, uqua, tmm n	0.5 NULL	≥ 5	0.2	0.02

4.3.1 NOISeq-real: using available replicates

NOISeq-real estimates the probability distribution for M and D in an empirical way, by computing M and D values for every pair of replicates within the same experimental condition and for every feature. Then, all these values are pooled together to generate the noise distribution. Two replicates in one of the experimental conditions are enough to run the algorithm. If the number of possible comparisons within a certain condition is higher than 30, in order to reduce computation time, 30 pairwise comparisons are randomly chosen when estimating noise distribution.

It should be noted that biological replicates are necessary if the goal is to make any inference about the population. Deriving differential expression from technical replicates is useful for drawing conclusions about the specific samples being compared in the study but not for extending these conclusions to the whole population.

In RNA-seq or similar sequencing technologies, the counts from technical replicates (e.g. lanes) can be summed up. Thus, this is the way the algorithm summarizes the information from technical replicates to compute M and D signal values (between different conditions). However, for biological replicates, other summary statistics such us the mean may be more meaningful. NOISeq calculates the mean of the biological replicates but we recommend to use NOISeqBIO when having biological replicates.

Here there is an example with technical replicates and count data normalized by `rpkm` method. Please note that, since the factor "Tissue" has two levels, we do not need to indicate which conditions are to be compared.

```
> mynoiseq = noiseq(mydata, k = 0.5, norm = "rpkm", factor = "Tissue", pnr = 0.2,
+     nss = 5, v = 0.02, lc = 1, replicates = "technical")

[1] "Computing (M,D) values..."
[1] "Computing probability of differential expression..."

> head(mynoiseq@results[[1]])

                Kidney_mean Liver_mean     M      D  prob ranking Length   GC Chrom GeneStart
ENSG00000177757       1.448      0.493 1.553  0.955 0.621    1.82   2464 48.6     1    742614
ENSG00000187634      19.860     11.706 0.763  8.154 0.751    8.19   4985 66.0     1    850393
ENSG00000188976      42.631     24.290 0.812 18.341 0.787   18.36   3870 59.5     1    869459
ENSG00000187961       6.289      5.225 0.267  1.064 0.428    1.10   4964 67.9     1    885830
ENSG00000187583       0.419      0.143 1.553  0.276 0.401    1.58   8507 62.6     1    891740
ENSG00000187642       2.524      0.412 2.616  2.113 0.782    3.36   6890 67.7     1    900447
                GeneEnd        Biotype
ENSG00000177757  745077        lincRNA
ENSG00000187634  869824 protein_coding
```

```
ENSG00000188976  884494 protein_coding
ENSG00000187961  890958 protein_coding
ENSG00000187583  900339 protein_coding
ENSG00000187642  907336 protein_coding
```

NA values would be returned if the gene had 0 counts in all the samples. In that case, the gene would not be used to compute differential expression.

Now imagine you want to compare tissues within the same sequencing run. Then, see the following example on how to apply NOISeq on count data with technical replicates, TMM normalization, and no length correction. As "TissueRun" has more than two levels we have to indicate which levels (conditions) are to be compared:

```
> mynoiseq.tmm = noiseq(mydata, k = 0.5, norm = "tmm", factor = "TissueRun",
+     conditions = c("Kidney_1", "Liver_1"), lc = 0, replicates = "technical")
```

4.3.2 NOISeq-sim: no replicates available

When there are no replicates available for any of the experimental conditions, NOISeq can simulate technical replicates. The simulation relies on the assumption that read counts follow a multinomial distribution, where probabilities for each class (feature) in the multinomial distribution are the probability of a read to map to that feature. These mapping probabilities are approximated by using counts in the only sample of the corresponding experimental condition. Counts equal to zero are replaced with k>0 to give all features some chance to appear.

Given the sequencing depth (total amount of reads) of the unique available sample, the size of the simulated samples is a percentage (parameter *pnr*) of this sequencing depth, allowing a small variability (given by the parameter v). The number of replicates to be simulated is provided by *nss* parameter.

Our dataset do has replicates but, providing it had not, you would use NOISeq-sim as in the following example in which the simulation parameters have to be chosen (*pnr*, *nss* and v):

```
> myresults <- noiseq(mydata, factor = "Tissue", k = NULL, norm = "n", pnr = 0.2,
+     nss = 5, v = 0.02, lc = 1, replicates = "no")
```

4.3.3 NOISeqBIO

NOISeqBIO is optimized for the use on biological replicates (at least 2 per condition). It was developed by joining the philosophy of our previous work together with the ideas from Efron *et al.* in [6]. In our case, we defined the differential expression statistic θ as $(M + D)/2$, where M and D are the statistics defined in the previous section but including a correction for the biological variability of the corresponding feature. The probability distribution of θ can be described as a mixture of two distributions: one for features changing between conditions and the other for invariant features. Thus, the mixture distribution f can be written as: $f(\theta) = p_0 f_0(\theta) + p_1 f_1(\theta)$, where p_0 is the probability for a feature to have the same expression in both conditions and $p_1 = 1 - p_0$ is the probability for a feature to have different expression between conditions. f_0 and f_1 are, respectively, the densities of θ for features with no change in expression between conditions and for differentially expressed features. If one of both distributions can be estimated, the probability of a feature to belong to one of the two groups can be calculated. Thus, the algorithm consists of the following steps:

1. Computing θ values.
   M and D are corrected for the biological variability: $M^* = \dfrac{M}{a_0 + \hat{\sigma}_M}$ and $D^* = \dfrac{D_s}{a_0 + \hat{\sigma}_D}$, where $\hat{\sigma}^2_M$ and $\hat{\sigma}^2_D$ are the standard errors of M_s and D_s statistics, respectively, and a_0 is computed as a given percentile of all the values in $\hat{\sigma}_M$ or $\hat{\sigma}_D$, as in [6] (the authors suggest the percentile 90th as the best option, which is the default option of the parameter "a0per" that may be changed by the user). To compute the θ statistic, the M and D statistics are combined: $\theta = \dfrac{M^* + D^*}{2}$.

2. Estimating the values of the θ statistic when there is no change in expression, i.e. the null statistic θ_0.
   In order to compute the null density f_0 afterwards, we first need to estimate the values of the θ-scores for features with no change between conditions. To do that, we permute r times (parameter that may be set by the user) the labels of samples between conditions, compute θ values as above and pool them to obtain θ_0.

3. Estimating the probability density functions f and f_0.
   We estimate f and f_0 with a kernel density estimator (KDE) with Gaussian kernel and smoothing parameter "adj" as indicated by the user.

4. Computing the probability of differential expression given the ratio f_0/f and an estimation $\hat{p}_0$ for p_0. If $\theta = z$ for a given feature, this probability of differential expression can be computed as $p_1(z) = 1 - \hat{p}_0 f_0(z)/f(z)$. To estimate p_0, the following upper bound is taken, as suggested in [6]: $p_0 \leq \min_Z\{f(Z)/f_0(Z)\}$. Moreover, it is shown in [6] that the FDR defined by Benjamini and Hochberg can be considered equivalent to the *a posteriori* probability $p_0(z) = 1 - p_1(z)$ we are calculating.

When too few replicates are available for each condition, the null distribution is very poor since the number of different permutations is low. For those cases (number of replicates per condition less than 5), it is convenient to borrow information across genes. Our proposal consists of clustering all genes according to their expression values across replicates using the k-means method. For each cluster k of genes, we considered the expression values of all the genes in the cluster as observations within the corresponding condition (replicates) and then we shuffled this submatrix $r \times g_k$ times, where g_k is the number of genes within cluster k. For each permutation, we calculated M and D values and their corresponding standard errors. In order to reduce the computing time, if $g_k \geq 1000$, we again subdivided cluster k in subclusters with k-means algorithm.

We will consider that Marioni's data have biological replicates for the following example. In this case, the method 2 (Wilcoxon test) to filter low counts is used. Please, use `?noiseqbio` to know more about the parameters of the function.

```
> mynoiseqbio = noiseqbio(mydata, k = 0.5, norm = "rpkm", factor = "Tissue",
+     lc = 1, r = 20, adj = 1.5, plot = FALSE, a0per = 0.9, random.seed = 12345,
+     filter = 2)
```

4.4 Results

4.4.1 Output object

NOISeq returns an `Output` object containing the following elements:

- `comparison`: String indicating the two experimental conditions being compared and the sense of the comparison.
- `factor`: String indicating the factor chosen to compute the differential expression.
- `k`: Value to replace zeros in order to avoid indetermination when computing logarithms.
- `lc`: Correction factor for length normalization. Counts are divided by $length^{lc}$.
- `method`: Normalization method chosen.
- `replicates`: Type of replicates: "technical" for technical replicates and "biological" for biological ones.
- `results`: R data frame containing the differential expression results, where each row corresponds to a feature. The columns are: Expression values for each condition to be used by `NOISeq` or `NOISeqBIO` (the columns names are the levels of the factor); differential expression statistics (columns"M" and "D" for `NOISeq` or "theta" for `NOISeqBIO`); probability of differential expression ("prob"); "ranking", which is a summary statistic of "M" and "D" values equal to $-sign(M) \times \sqrt{M^2 + D^2}$, than can be used for instance in gene set enrichment analysis (only for `NOISeq`); "Length" of each feature (if provided); "GC" content of each feature (if provided); chromosome where the feature is ("Chrom"), if provided; start and end position of the feature within the chromosome ("GeneStart", "GeneEnd"), if provided; feature biotype ("Biotype"), if provided.
- `nss`: Number of samples to be simulated for each condition (only when there are not replicates available).
- `pnr`: Percentage of the total sequencing depth to be used in each simulated replicate (only when there are not replicates available). For instance, if pnr = 0.2 , each simulated replicate will have 20% of the total reads of the only available replicate in that condition.
- `v`: Variability of the size of each simulated replicate (only used by NOISeq-sim).

For example, you can use the following instruction to see the differential expression results:

```
> head(mynoiseq@results[[1]])
```

```
                Kidney_mean Liver_mean     M      D  prob ranking Length   GC Chrom GeneStart
ENSG00000177757       1.448      0.493 1.553  0.955 0.621    1.82   2464 48.6     1    742614
ENSG00000187634      19.860     11.706 0.763  8.154 0.751    8.19   4985 66.0     1    850393
ENSG00000188976      42.631     24.290 0.812 18.341 0.787   18.36   3870 59.5     1    869459
ENSG00000187961       6.289      5.225 0.267  1.064 0.428    1.10   4964 67.9     1    885830
ENSG00000187583       0.419      0.143 1.553  0.276 0.401    1.58   8507 62.6     1    891740
ENSG00000187642       2.524      0.412 2.616  2.113 0.782    3.36   6890 67.7     1    900447
                GeneEnd        Biotype
ENSG00000177757  745077        lincRNA
ENSG00000187634  869824 protein_coding
ENSG00000188976  884494 protein_coding
ENSG00000187961  890958 protein_coding
ENSG00000187583  900339 protein_coding
ENSG00000187642  907336 protein_coding
```

The output `myresults@results[[1]]$prob` gives the estimated probability of differential expression for each feature. Note that when using NOISeq, these probabilities are not equivalent to p-values. The higher the probability, the more likely that the difference in expression is due to the change in the experimental condition and not to chance. See Section 4.4 to learn how to obtain the differentially expressed features.

4.4.2 How to select the differentially expressed features

Once we have obtained the differential expression probability for each one of the features by using `NOISeq` or `NOISeqBIO` function, we may want to select the differentially expressed features for a given threshold q. This can be done with `degenes` function on the "output" object using the parameter `q`. With the argument `M` we choose if we want all the differentially expressed features, only the differentially expressed features that are more expressed in condition 1 than in condition 2 (M = "up") or only the differentially expressed features that are under-expressed in condition 1 with regard to condition 2 (M = "down"):

```
> mynoiseq.deg = degenes(mynoiseq, q = 0.8, M = NULL)

[1] "1614 differentially expressed features"

> mynoiseq.deg1 = degenes(mynoiseq, q = 0.8, M = "up")

[1] "1289 differentially expressed features (up in first condition)"

> mynoiseq.deg2 = degenes(mynoiseq, q = 0.8, M = "down")

[1] "325 differentially expressed features (down in first condition)"
```

Please remember that, when using `NOISeq`, the probability of differential expression is not equivalent to $1 - pvalue$. We recommend for q to use values around 0.8. If `NOISeq-sim` has been used because no replicates are available, then it is preferable to use a higher threshold such as $q = 0.9$. However, when using `NOISeqBIO`, the probability of differential expression would be equivalent to $1 - FDR$, where FDR can be considered as an adjusted p-value. Hence, in this case, it would be more convenient to use $q = 0.95$.

4.4.3 Plots on differential expression results

Expression plot

Once differential expression has been computed, it is interesting to plot the expression values in each condition and highlight the features declared as differentially expressed. It can be done with the `DE.plot`.

To plot the summary of the expression values in both conditions as in Fig. 10, please write the following code (many graphical parameters can be adjusted, see the function help). Note that by giving $q = 0.9$, differentially expressed features considering this threshold will be highlighted in red:

```
> DE.plot(mynoiseq, q = 0.9, graphic = "expr", log.scale = TRUE)

[1] "888 differentially expressed features"
```

MD plot

Instead of plotting the expression values, one may be interested in plotting the (M, D) values as in Fig. 11. Remember that M is the log-fold-change and D is the absolute value of the difference between conditions. This is an example of the code to get such a plot (D values are displayed in log-scale).

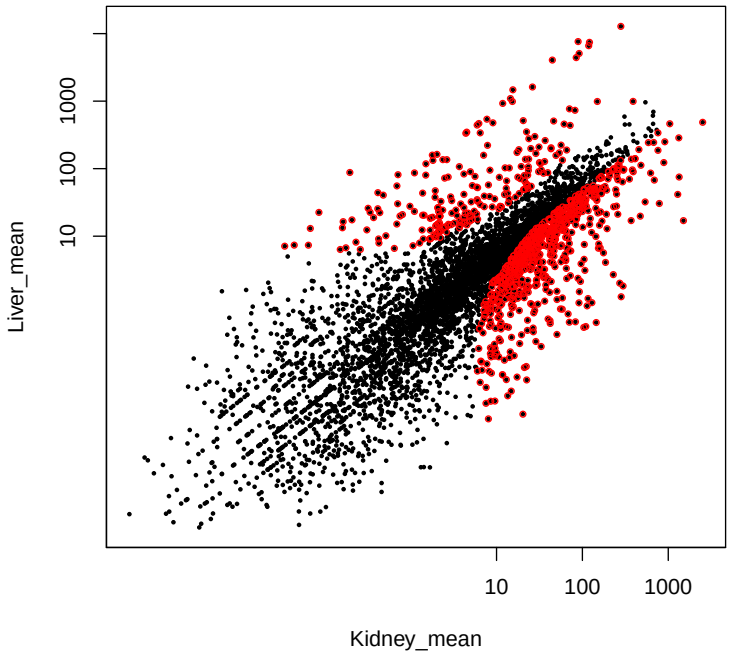

Figure 10: Summary plot of the expression values for both conditions (black), where differentially expressed genes are highlighted (red).

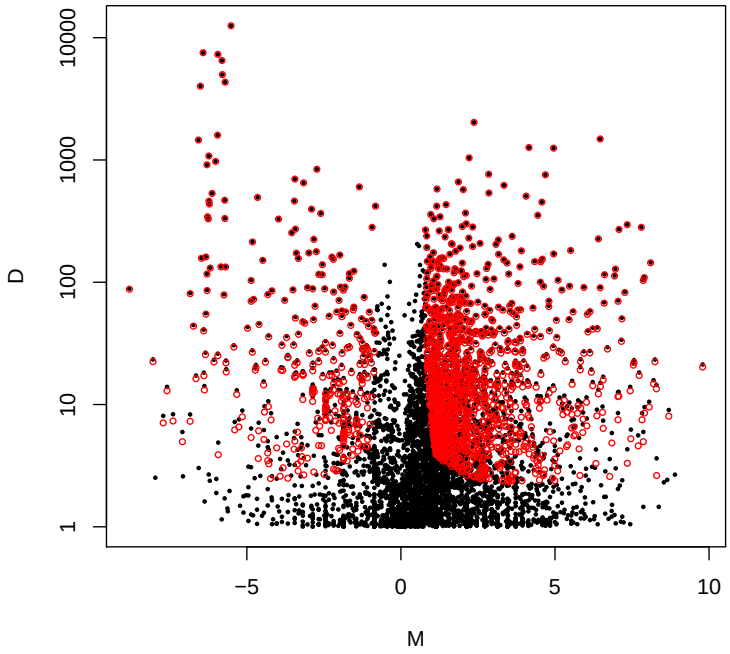

Figure 11: Summary plot for (M,D) values (black) and the differentially expressed genes (red).

```
> DE.plot(mynoiseq, q = 0.8, graphic = "MD")

[1] "1614 differentially expressed features"
```

Manhattan plot

The Manhattan plot can be used to display the features expression across the chromosomes. Expression for the two conditions under comparison is shown in the same plot. Users may choose either plotting all the chromosomes or only some of them, and also if the chromosomes are depicted consecutively (useful for prokaryote organisms) or separately (one per line). If a q cutoff is provided, then differentially expressed features are highlighted in a different color. The following code shows how to draw the Manhattan plot from the output object returned by `NOISeq` or `NOISeqBIO`. In this case, using Marioni's data, the log-expression is represented for two chromosomes (see Fig. 12). Note that the chromosomes will be depicted in the same order that are given to "chromosomes" parameter.

Gene expression is represented in gray. Lines above 0 correspond to the first condition under comparison (kidney) and lines below 0 are for the second condition (liver). Genes up-regulated in the first condition are highlighted in red, while genes up-regulated in the second condition are highlighted in green. The blue lines on the horizontal axis (Y=0) correspond to the annotated genes. X scale shows the location in the chromosome.

```
> DE.plot(mynoiseq, chromosomes = c(1, 2), log.scale = TRUE, join = FALSE,
+     q = 0.8, graphic = "chrom")
```

```
[1] "REMEMBER. You are plotting these chromosomes and in this order:"
[1] 1 2
[1] "1289 differentially expressed features (up in first condition)"
[1] "325 differentially expressed features (down in first condition)"
```

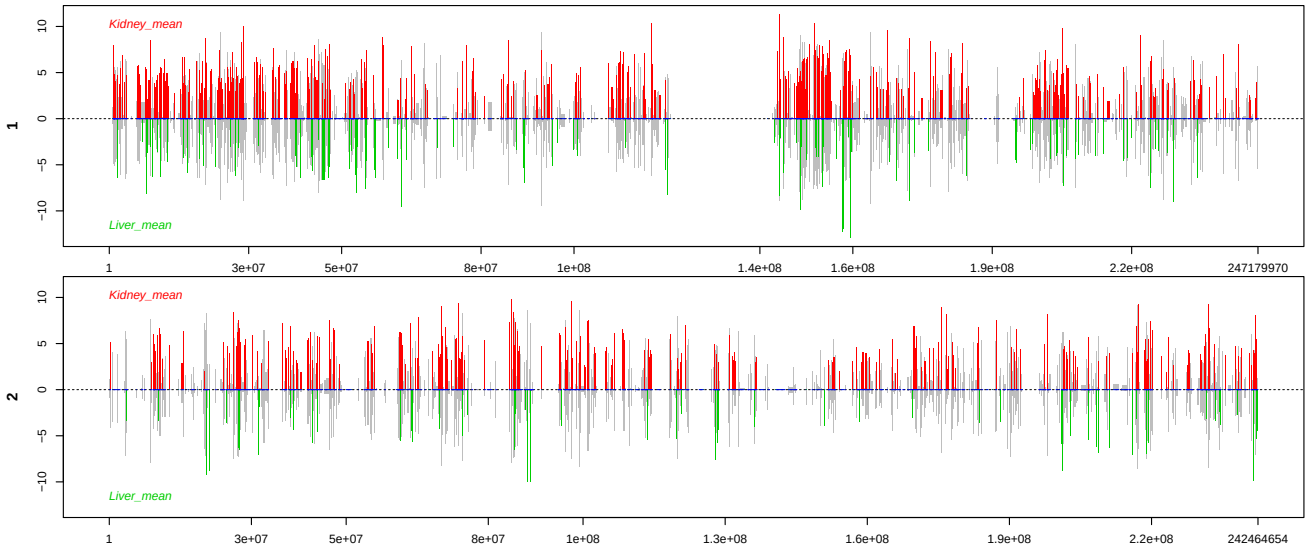

Figure 12: Manhattan plot for chromosomes 1 and 2

It is advisable, in this kind of plots, to save the figure in a file, for instance, a pdf file (as in the following code), in order to get a better visualization with the zoom.

```
pdf("manhattan.pdf", width = 12, height = 50)
DE.plot(mynoiseq, chromosomes = c(1,2), log.scale = TRUE,
        join = FALSE, q = 0.8)
dev.off()
```

Distribution of differentially expressed features per chromosomes or biotypes

This function creates a figure with two plots if both chromosomes and biotypes information is provided. Otherwise, only a plot is depicted with either the chromosomes or biotypes (if information of any of them is available). The q cutoff must be provided. Both plots are analogous. The chromosomes plot shows the percentage of features in each chromosome, the proportion of them that are differentially expressed (DEG) and the percentage of differentially expressed features in each chromosome. Users may choose plotting all the chromosomes or only some of them. The chromosomes are depicted according to the number of features they contain (from the greatest to the lowest). The plot for biotypes can be described similarly. The only difference is that this plot has a left axis scale for the most abundant biotypes and a right axis scale for the rest of biotypes, which are separated by a green vertical line.

The following code shows how to draw the figure from the output object returned by `NOISeq` for the Marioni's example data.

```
> DE.plot(mynoiseq, chromosomes = NULL, q = 0.8, graphic = "distr")

[1] "1614 differentially expressed features"
```

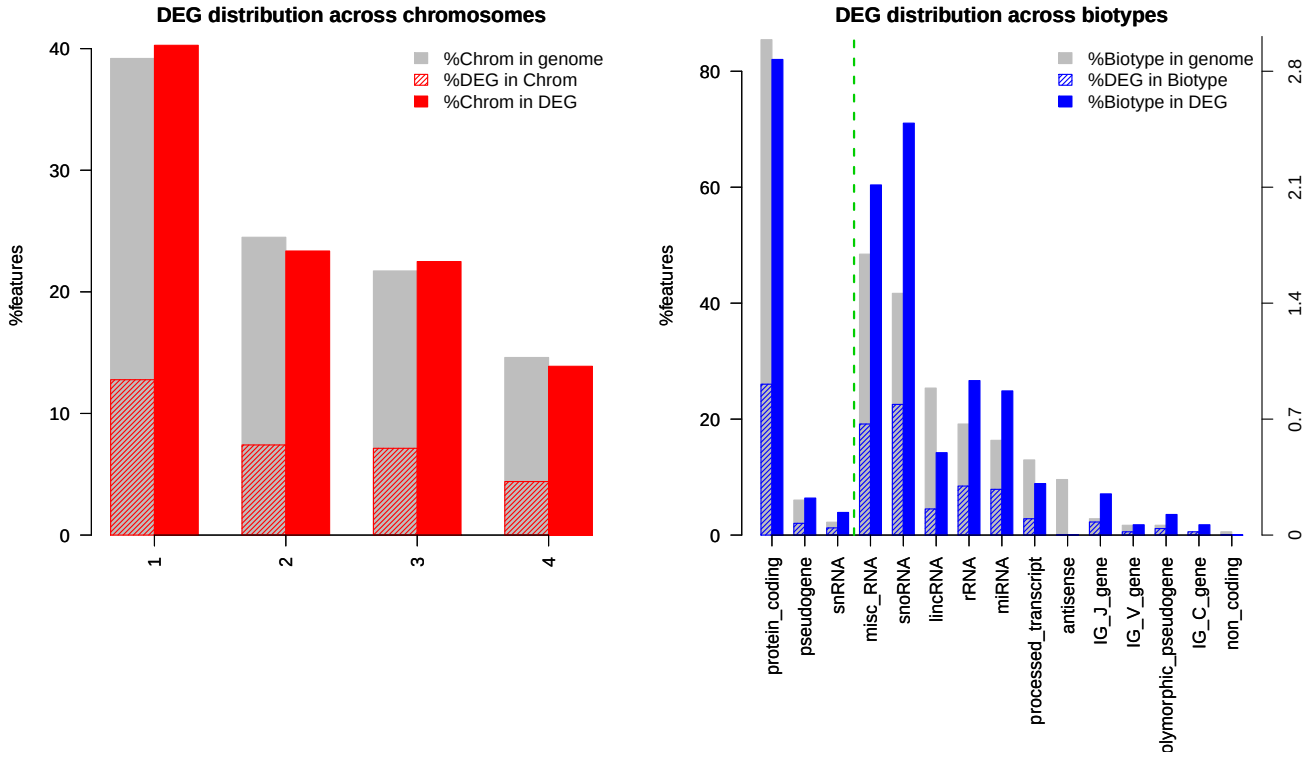

Figure 13: Distribution of DEG across chromosomes and biotypes for Marioni's example dataset.

5 Setup

This vignette was built on:

```
> sessionInfo()

R version 3.1.0 (2014-04-10)
Platform: x86_64-unknown-linux-gnu (64-bit)

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

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

other attached packages:
[1] NOISeq_2.6.0        Biobase_2.24.0      BiocGenerics_0.10.0

loaded via a namespace (and not attached):
[1] tools_3.1.0
```

References

[1] S. Tarazona, F. García-Alcalde, J. Dopazo, A. Ferrer, and A. Conesa. Differential expression in RNA-seq: A matter of depth. *Genome Research*, 21: 2213 - 2223, 2011.

[2] J.C. Marioni, C.E. Mason, S.M. Mane, M. Stephens, and Y. Gilad. RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research*, 18: 1509 - 517, 2008.

[3] A. Mortazavi, B.A. Williams, K. McCue, L. Schaeffer, and B. Wold. Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods*, 5: 621 - 628, 2008.

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

[5] M.D. Robinson, and A. Oshlack. A scaling normalization method for differential expression analysis of RNA-Seq data. *Genome Biology*, 11: R25, 2010.

[6] B. Efron, R. Tibshirani, J.D. Storey, V. Tusher. Empirical Bayes Analysis of a Microarray Experiment. *Journal of the American Statistical Association.*