NA

NOISeq: Differential Expression in RNA-seq

28 January 2013
(Version 1.1.5)

Contents

1 Introduction

This document is intended to guide you through the use of the R Bioconductor package NOISeq, to analyze count data coming from next generation sequencing technologies. First, we describe the input data format. Then, we suggest you to explore your 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 with NOISeq. The differential expression method NOISeq and some of the plots included in the package were used and described in [1].

NOISeq differs from existing methods in that it is data-adaptive and nonparametric. Other methodologies may suffer from a strong dependency on sequencing depth for their differential expression calls and this results in a considerable number of false positives that increases as the number of reads grows. In contrast, as it is shown in [1], NOISeq models the noise distribution from the actual data, can therefore better adapt to the size of the data set, and is more effective in controlling the rate of false discoveries.

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 some of the exploratory plots included in the package, other additional annotations must be provided such as the feature length, 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, with so many rows as the number of features to be studied and so many 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
ENSG00000197049	0	1	0	0	0	0	0
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
	R2L2Kidney	R2L3Liver	R2L6Kidney				
ENSG00000197049	1	0	0				
ENSG00000177757	1	1	3				
ENSG00000187634	42	25	47				
ENSG00000188976	70	42	82				
ENSG00000187961	12	20	15				
ENSG00000187583	0	2	3				

The expression data may be the read counts but also normalized expression data such as RPKM values or any other normalization.

2.2 Factors

Factors are given in a data.frame object like follows. For Marioni's data, we obviously have the factor "Tissue", but we will also define another toy factor ("TissueRun") just to show what 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 then has four levels: "Kidney_1", "Liver_1", "Kidney_2" and "Liver_2".

Be careful here, the order of the levels of the factor must coincide with the order of the samples 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 of 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 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 must look like:

```
> head(mylength)

ENSG00000197049 ENSG00000177757 ENSG00000187634 ENSG00000188976 ENSG00000187961
           1194            2464            4985            3870            4964
ENSG00000187583
           8507

> head(mybiotypes)

ENSG00000197049 ENSG00000177757 ENSG00000187634 ENSG00000188976 ENSG00000187961
 protein_coding         lincRNA  protein_coding  protein_coding  protein_coding
ENSG00000187583
 protein_coding
16 Levels: antisense IG_C_gene IG_J_gene IG_V_gene lincRNA miRNA misc_RNA ... snRNA

> head(mychroms)

                Chr GeneStart GeneEnd
ENSG00000197049   1    711269  711775
ENSG00000177757   1    742614  745077
ENSG00000187634   1    850393  869824
ENSG00000188976   1    869459  884494
ENSG00000187961   1    885830  890958
ENSG00000187583   1    891740  900339
```

Please note, that these objects may not contain the same number of features or in the same order than the expression data. But it is important to especify the names of the features in each case. For the length or the biological groups (e.g. biotypes), which are vectors, the names of the vector must be the feature names or ids. In the case of the chromosome information, which is a matrix or data.frame, the feature names or ids must be in the row names of the object.

2.4 Usage

To read these data and 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`, `biotype`, `chromosome` and `factors` receive the objects shown before:

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

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 5105 features, 10 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: R1L1Kidney R1L2Liver ... R2L6Kidney (10 total)
  varLabels: Tissue TissueRun
  varMetadata: labelDescription
featureData
  featureNames: ENSG00000197049 ENSG00000177757 ... ENSG00000201145 (5105 total)
  fvarLabels: Length Biotype ... GeneEnd (5 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. But, in that case, perhaps no annotations are 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)
```

3 Exploratory analysis

In this section we present some plots that may be useful to explore the count data, specially when a biological classification of the features is provided. For instance, if the features to be studied are the genes, it is interesting to see which kind of genes are being detected in our RNA-seq samples, how many counts are they being detected with, etc. These plots can also be used to evaluate if there are differences among chromosomes or among features of different length,...

Please note that although `readData()` accepts any kind of expression data (not only read counts but also normalized data), the plots in this section may be not meaningful for expression data different from read counts.

3.1 Generating data for exploratory plots

First of all, `dat` function must be applied on the input data to obtain the results to be plotted by *explo.plot*. Since there are five types of exploratory plots that can be obtained with *explo.plot*, the *explo.plot* function needs the argument `type` to know which type of data is to be generated. For instance:

```
> myexplodata <- dat(mydata, type = "biodetection", selection = c(3, 4))
```

In the following sections you will find more details about each type of exploratory plots.

3.2 Biodetection

When the biological classification of the features is provided (e.g. biotypes), a bar diagram can be obtained that 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`) and the percentage of each biotype within the sample.

This is an example on how to use the corresponding functions, first to generate the data to be plotted and then to draw the plot. 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, selection = c(1, 2), k = 0, type = "biodetection")
> explo.plot(mybiodetection)
```

Fig. 1 shows the plot that would be generated. 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).

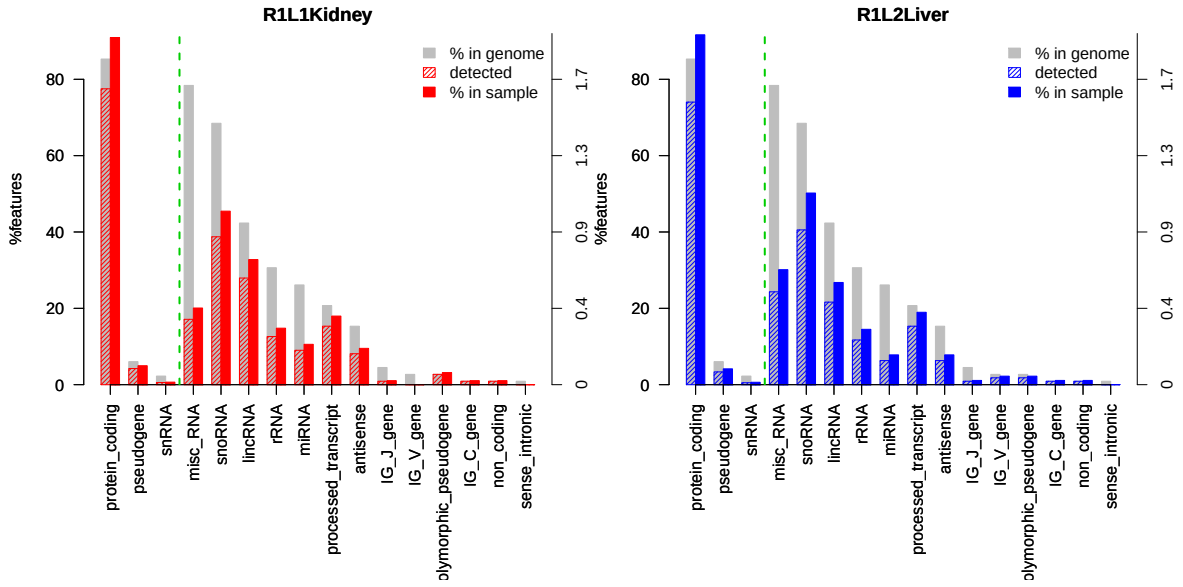

Figure 1: Biodetection plot

3.3 Count distribution comparison

This plot is to check if the distribution of the counts is the same for two given conditions, which would be desirable in order to compute differential expression between them. See an usage example below to compare samples 1 and 2, and the result in Fig. 2:

```
> mycd = dat(mydata, selection = c(1, 2), type = "cd")
> explo.plot(mycd)
```

In the plot, it can be seen that the distributions are different for kidney and liver samples. For example, in kidney 20% of the genes account for about 65% of the reads, while in liver they account for more than 80% of the reads.

3.4 Distribution of counts per biological group

With this plot, and providing that we have a biological classification for the features (e.g. biotype), we can see how the counts are distributed within each biological group.

The following code generates the data for a plot of type "countsbio" from all the samples (as `selection` parameter is NULL the results are computed for all the samples). The `ndepth` parameter is indicating that the distribution of counts will be computed at 5 increasing sequencing depths. The last one will be the real sequencing depth and the rest of them will be simulated by sampling from the total number of reads according the multinomial probability distribution. With *explo.plot*, a plot is generated that shows the distribution of counts per biotype for samples 1 and 2 and for all the biotypes (because `toplot` parameter is 1) as the one in Fig. 3.

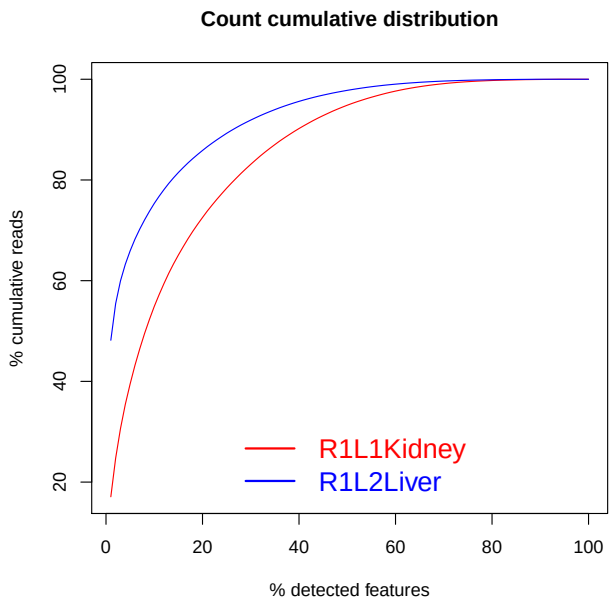

Figure 2: Count distribution plot

```
> mycountsbio = dat(mydata, selection = NULL, k = 0, ndepth = 5, type = "countsbio")
> explo.plot(mycountsbio, toplot = 1, samples = c(1, 2), ylim = c(0, 200))
```

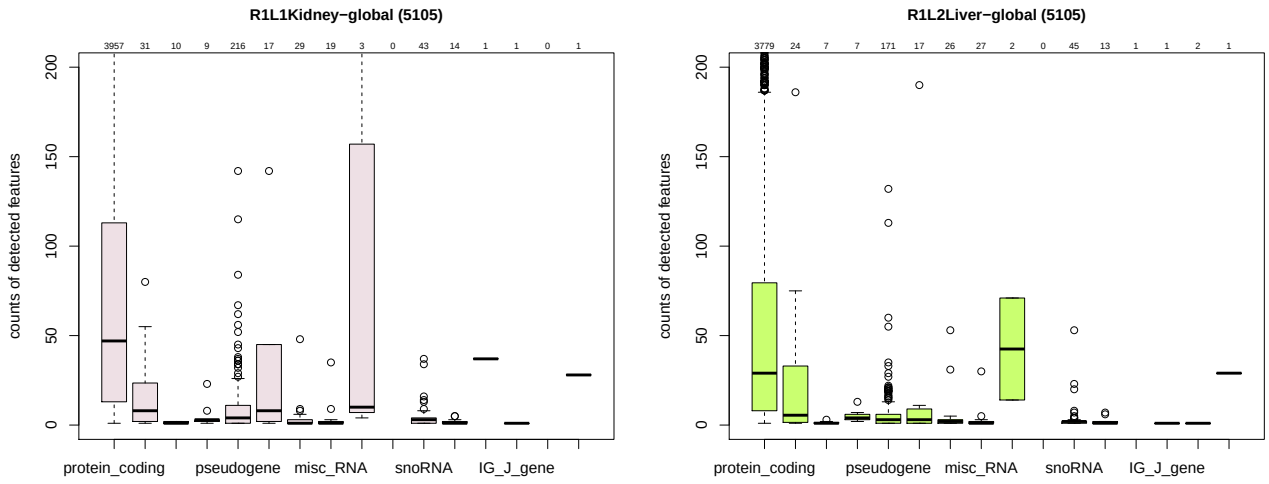

Figure 3: Count distribution per biotype in two samples (for genes with more than 0 counts). At the upper part of the plot, the number of detected features within each biotype group is displayed.

To see how the distribution of counts changes when increasing sequencing depth, the following code will generate the example in Fig. 4 for the biotype "protein_coding" and the sample number 5.

```
> explo.plot(mycountsbio, toplot = "protein_coding", samples = 5, ylim = c(0,
+     200))
```

3.5 Length of detected features

The plot in Fig. 5 is to describe the median length of detected features within each sample and per biotype when increasing sequencing depth. In this example, we plotted 4 samples (from 3 to 6) and the "protein_coding"

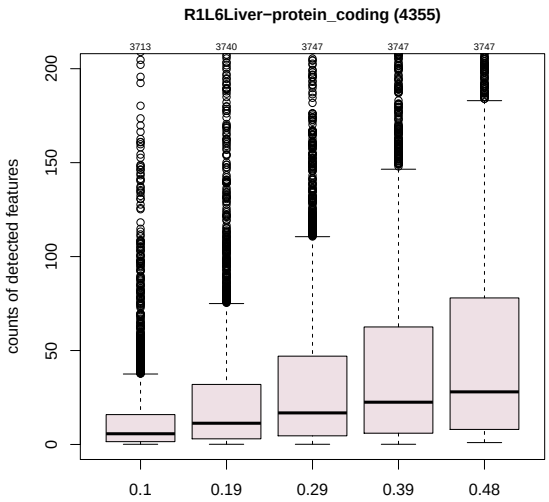

Figure 4: Distribution of counts for protein coding genes in one of the samples when increasing sequencing depth.

biotype. First we compute the values for all the samples and biotypes and then we plot the chosen samples and/or biotypes.

```
> myDLbio = dat(mydata, selection = NULL, k = 0, ndepth = 5, type = "DLbio")
> explo.plot(myDLbio, samples = 3:6, toplot = "protein_coding")
```

3.6 Saturation plots

Sometimes, it is interesting to know how many of the features in the genome we are detecting with more than a given number of counts (e.g. k=5). If we can see how this number of detected features changes when increasing sequencing depth, it may give us an idea about the saturation in our sample: should we increase our sequencing depth or if we increase it we are not detecting many more new features? The saturation plots allow you to study this phenomenon for each biotypes (if you provide them) or for all the genes in general.

First, we have to generate the saturation data with the function `dat` and then we can use the resulting data to obtain, for instance, the plots in Fig. 6 and 7 by using *explo.plot*. When `newdetections` parameter is set to TRUE, 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, this parameter is automatically set to FALSE because it is difficult to visualize the newdetection bars with so many samples.

```
> mysaturation = dat(mydata, k = 0, ndepth = 5, newdetections = TRUE, type = "saturation")
> explo.plot(mysaturation, toplot = 1, samples = 1:2, ylim = NULL, yrightlim = NULL)

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

The plot in Fig. 6 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 right end of the lines corresponds to the total 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.

It is possible to compare more than 2 samples, but then the information in the right axis is not provided because it would be difficult to distinguish so many bars. You can compare up to twelve samples. In Fig. 7, 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.

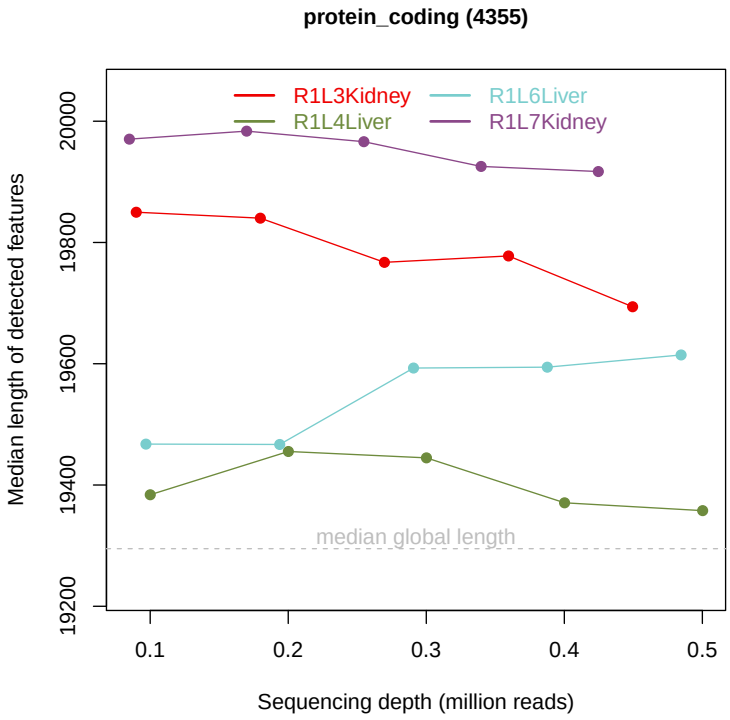

Figure 5: Median length of detected features at increasing sequencing depths.

4 Differential expression

The NOISeq method computes differential expression between two experimental conditions given the expression level of the considered features. The method is intended to use read counts from RNA-seq as the expression values.

We strongly recommend to normalize the counts to correct, at least, sequencing depth bias, and also feature length bias, if possible. There are some parameters in NOISeq to indicate the method how to normalize counts. See Section 4.1 and Section 4.2 to get more information about normalization.

NOISeq takes the normalized counts to obtain the statistics needed to derive differential expression. First, if there are replicates available, it summarizes the replicates information by summing up the technical replicates or by computing the average of biological replicates. Next, NOISeq computes the differential expression statistics for each feature: M (which is the log_2-ratio of the two conditions) and D (the absolute 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 note 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 your results when having no replicates, since they are not showing real differential expression, but they are only an approximation to show you which genes are presenting a higher change between conditions.

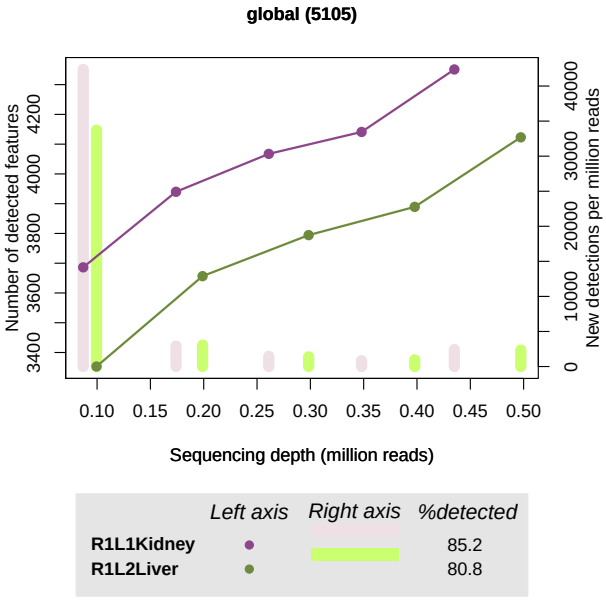

Figure 6: Global saturation plot to compare two samples of kidney and liver, respectively.

4.1 Input parameters

Depending on the nature of the available samples (biological, technical or no replicates), we have to choose the appropriate values for the parameters in the corresponding NOISeq. This is a short description for the input parameters:

- `input`: The `Input` object generated from the `readData` function, including at least one factor with exactly two levels.
- `replicates`: In this argument, the type of replicates to be used is defined. Technical, biological or none. By default, technical replicates option is chosen.
- `k`: Counts equal to 0 are replaced by k. By default, $k = 0.5$.
- `norm`: Normalization method. It can be one of "rpkm" (default), "uqua" (Upper Quartile), "tmm" (Trimmed Mean of M) or "n" (no normalization).
- `lc`: Correction factor for length normalization. This correction is done by dividing the counts vector by $length^{lc}$. By default, lc = 1.
- `factor`: A string indicating the name of the factor that defines the conditions to be compared.
- `conditions`: A vector containing the two conditions to be compared by the differential expression algorithm. It is needed when the factor has more than 2 levels (i.e. it contains more than 2 different conditions).
- `nss`: Number of samples to be simulated within each condition when there are no replicates available. By default, nss = 5.
- `pnr`: Percentage of the total reads in each condition for each simulated sample. To be specified only when there are not replicates available. By default, pnr = 0.2.
- `v`: Variability in the total sequencing depth of the simulated samples (only for NOISeq-sim). By default, v = 0.02.

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

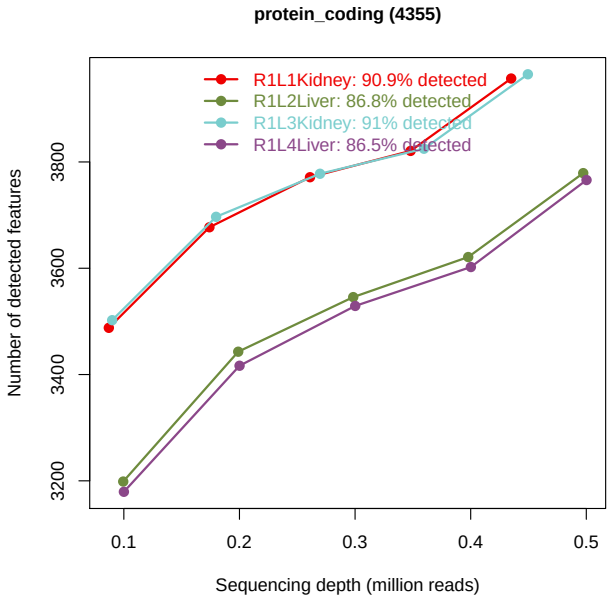

Figure 7: Saturation plot for protein-coding genes to compare 4 samples: 2 of kidney and 2 of liver.

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

4.2 More about normalization

The normalization techniques implemented in `NOISeq` are RPKM [3], Upper Quartile [4] and TMM, which stands for Trimmed Mean of M values [5]. NOISeq also accepts normalized expression values instead of counts in case the user would prefer to apply another normalization procedure.

If the length of the features is provided and any of the three normalization methods is chosen, the expression values are divided by $length^{lc}$ ($lc = 1$ 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.

Normalization functions can also be used independently of `NOISeq` differential expression computation. Please, find below some examples on how to do that:

```
> 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 = 1)
> head(myRPKM[, 1:4])
```

```
                R1L1Kidney R1L2Liver R1L3Kidney R1L4Liver
ENSG00000197049       0.00     1.684      0.000     0.000
ENSG00000177757       1.87     0.816      0.000     0.000
ENSG00000187634      22.60    10.891     19.191    13.635
ENSG00000188976      43.36    17.663     44.261    28.925
ENSG00000187961       6.95     3.240      6.723     5.235
ENSG00000187583       0.27     0.000      0.262     0.235
```

4.3 NOISeq-real: using available replicates

NOISeq-real estimates the probability distribution for M and D in an empirical way, 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 to draw conclusions about the specific samples being compared in the study but not to extend these conclusions to the whole population.

In RNA-seq or similar sequencing technologies, 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 or the median may be more meaningful. In this version of NOISeq, we have chosen to calculate the mean of the 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 Chrom GeneStart
ENSG00000197049       0.373      0.339 0.138  0.0341 0.126   0.142   1194     1    711269
ENSG00000177757       1.448      0.493 1.553  0.9544 0.621   1.823   2464     1    742614
ENSG00000187634      19.858     11.705 0.763  8.1527 0.751   8.188   4985     1    850393
ENSG00000188976      42.627     24.289 0.811 18.3379 0.786  18.356   3870     1    869459
ENSG00000187961       6.288      5.224 0.267  1.0636 0.428   1.097   4964     1    885830
ENSG00000187583       0.419      0.143 1.553  0.2764 0.401   1.578   8507     1    891740
                GeneEnd        Biotype
ENSG00000197049  711775 protein_coding
ENSG00000177757  745077        lincRNA
ENSG00000187634  869824 protein_coding
ENSG00000188976  884494 protein_coding
ENSG00000187961  890958 protein_coding
ENSG00000187583  900339 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")
```

4.4 NOISeq-sim: no replicates available

When there are no replicates available for any of the experimental conditions, the algorithm 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 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 of the unique available sample, the size of the simulated samples is a percentage (parameter *pnr*) of this total amount of reads, 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")
```

4.5 Output data

`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` (the columns names are the levels of the factor); differential expression statistics (columns "M" and "D"); 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; "length" 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). If, for instance, 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, to see the differential expression results, you can use the following instruction:

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

```
                Kidney_mean Liver_mean     M       D  prob ranking Length Chrom GeneStart
ENSG00000197049       0.373      0.339 0.138  0.0341 0.126   0.142   1194     1    711269
ENSG00000177757       1.448      0.493 1.553  0.9544 0.621   1.823   2464     1    742614
ENSG00000187634      19.858     11.705 0.763  8.1527 0.751   8.188   4985     1    850393
ENSG00000188976      42.627     24.289 0.811 18.3379 0.786  18.356   3870     1    869459
ENSG00000187961       6.288      5.224 0.267  1.0636 0.428   1.097   4964     1    885830
ENSG00000187583       0.419      0.143 1.553  0.2764 0.401   1.578   8507     1    891740
                GeneEnd        Biotype
ENSG00000197049  711775 protein_coding
ENSG00000177757  745077        lincRNA
ENSG00000187634  869824 protein_coding
ENSG00000188976  884494 protein_coding
ENSG00000187961  890958 protein_coding
ENSG00000187583  900339 protein_coding
```

The output `myresults@results[[1]]$prob` gives the estimated probability of differential expression for each feature. Note that these probabilities are not 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 4.6 to learn how to obtain the differentially expressed features.

4.6 How to select the differentially expressed features

We have obtained the differential expression probability of each gene by using `NOISeq` function. Please note that this probability is not equivalent to $1 - pvalue$. We recommend to select as differentially expressed genes those genes with a probability higher than $q = 0.8$. If NOISeq-sim has been used because no replicates are available, then it is preferable to use a higher threshold such as $q = 0.9$.

It is very simple to select the differentially expressed genes directly from the `NOISeq` output, but the `degenes` function may also be used. With the argument "M" we may choose if we want all the differentially expressed genes, only the differentially expressed genes that are more expressed in condition 1 than in condition 2 (M = "up") or only the differentially expressed genes that are under-expressed in condition 1 with regard to condition 2 (M = "down"):

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

[1] "1611 differentially expressed features"

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

[1] "1285 differentially expressed features (up in first condition)"

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

[1] "326 differentially expressed features (down in first condition)"
```

4.7 Plots on differential expression

4.7.1 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. 8, please write the following code (many graphical parameters can be adjusted, see the function help). Note that by giving $q = 0.8$, differentially expressed features considering this threshold will be highlighted in red:

```
> DE.plot(mynoiseq, q = 0.8, graphic = "expr", log.scale = TRUE)

[1] "1611 differentially expressed features"
```

4.7.2 (M, D) plot

Instead of plotting the expression values, one may be interested in plotting the (M, D) values as in Fig. 9. This is an example of the code to get such a plot (in this case we have zoomed to see the only the genes with D values smaller than 50):

```
> DE.plot(mynoiseq, q = 0.8, graphic = "MD", ylim = c(0, 50))

[1] "1611 differentially expressed features"
```

4.7.3 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 if 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`. In this case, using Marioni's data, the log-expression is represented for two chromosomes (see Fig. 10). 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.

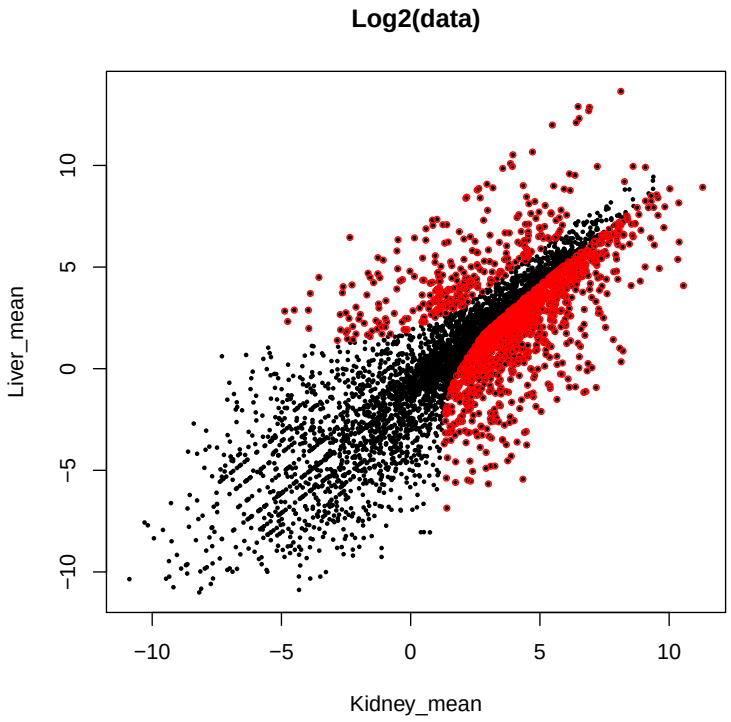

Figure 8: Summary plot of the expression values for both conditions (black), where differentially expressed genes are highlighted (red).

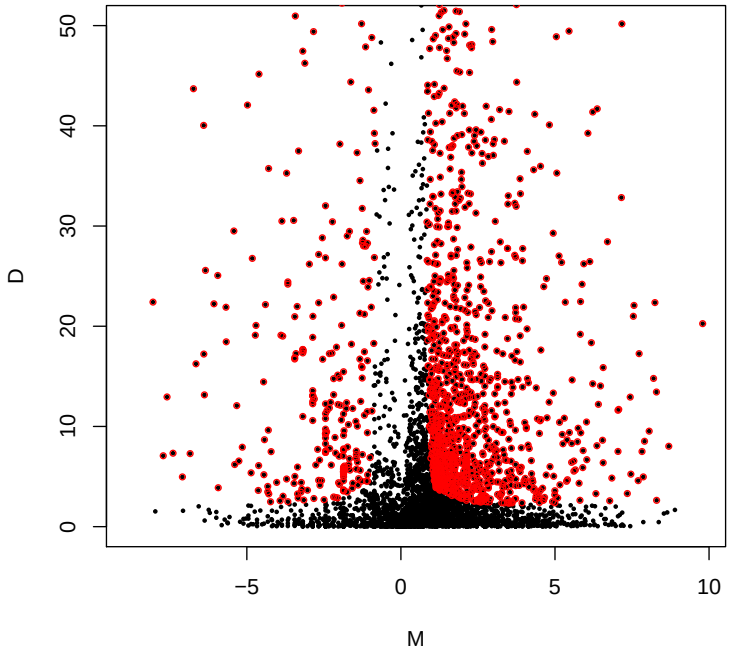

Figure 9: Summary plot for (M,D) values (black) and the differentially expressed genes (red).

```
> 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] "1285 differentially expressed features (up in first condition)"
[1] "326 differentially expressed features (down in first condition)"
```

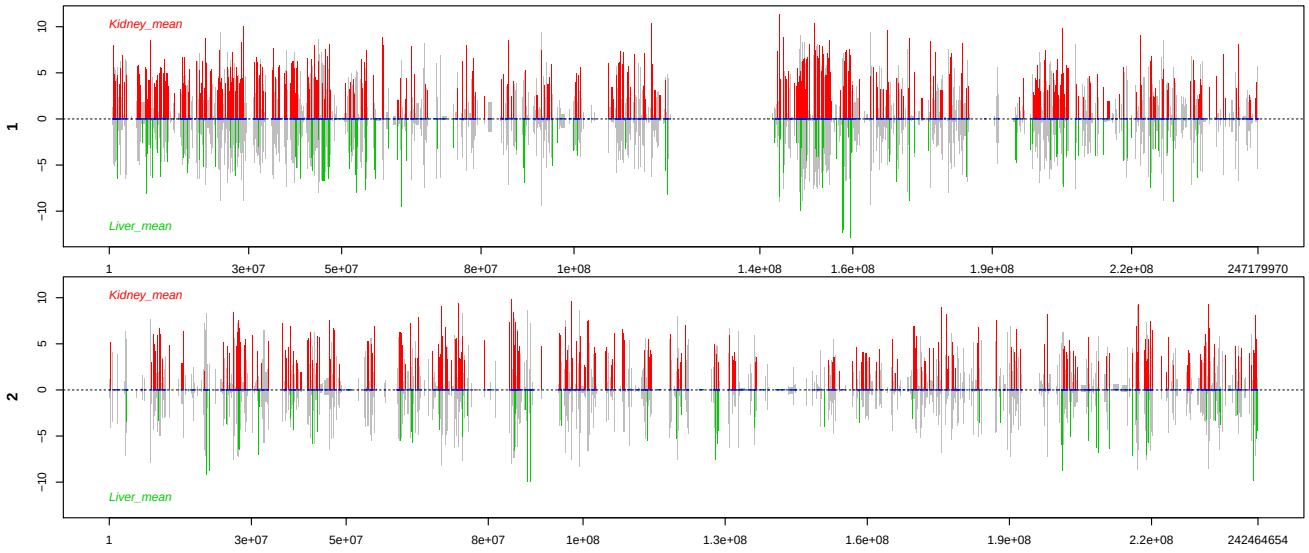

Figure 10: Manhattan plot for chromosomes 1 and 2

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()
```

4.7.4 Plot showing the distribution of differentially expressed features across 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] "1611 differentially expressed features"
```

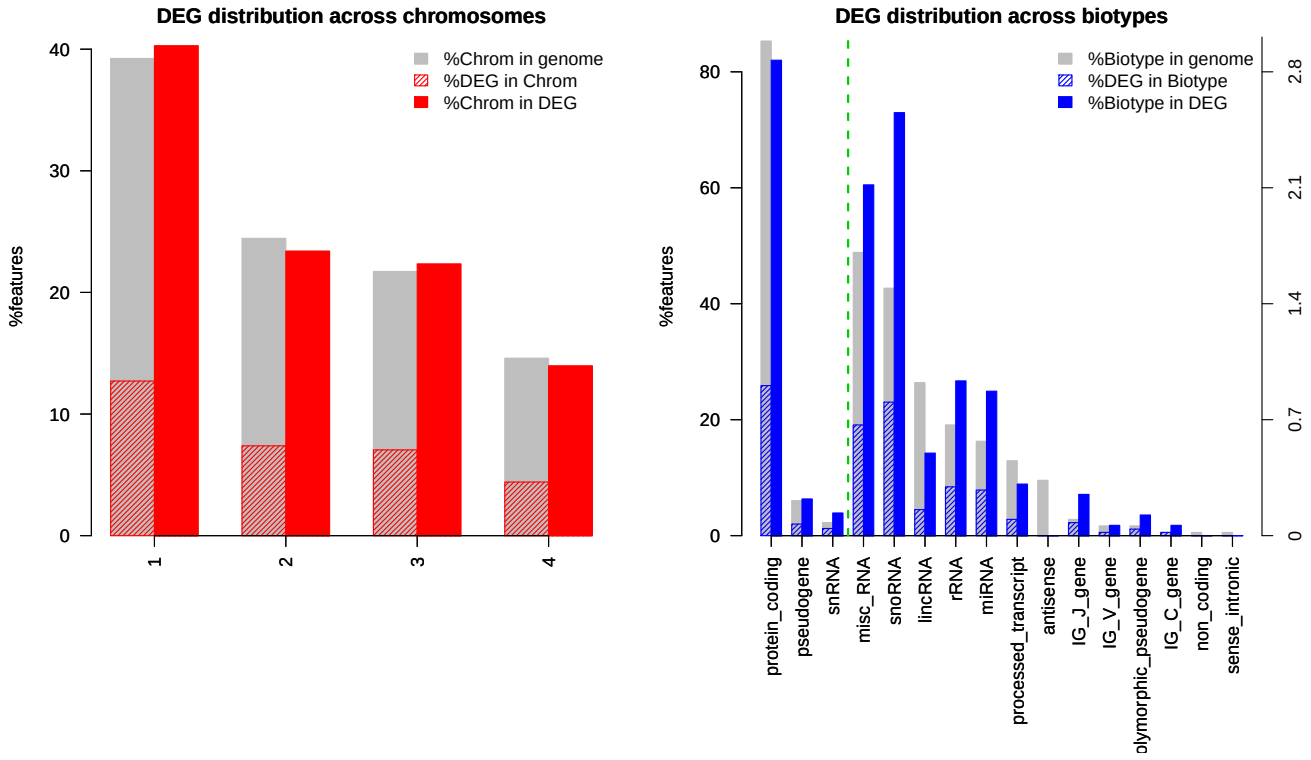

Figure 11: Distribution of DEG across chromosomes and biotypes for Marioni's example dataset.

5 Setup

This vignette was built on:

```
> sessionInfo()

R version 2.15.2 (2012-10-26)
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

locale:
[1] C/en_US.US-ASCII/en_US.US-ASCII/C/en_US.US-ASCII/en_US.US-ASCII

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

other attached packages:
[1] NOISeq_1.1.5       Biobase_2.18.0     BiocGenerics_0.4.0

loaded via a namespace (and not attached):
[1] tools_2.15.2
```

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. NA

[3] A. Mortazavi, B.A. Williams, K. McCue, L. Schaeffer, and B. Wold. Mapping and quantifying mammalian transcriptomes by RNA-Seq. NA

[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.