AgiMicroRna

Pedro Lopez-Romero

October 14, 2013

1 Package Overview

AgiMicroRna provides useful functionality for the processing, quality assessment and differential expression analysis of Agilent microRNA array data. The package uses a limma-like structure to generate the processed data in order to make statistical inferences about differential expression using the linear model features implemented in *limma*. Standard Bioconductor objects are used so that other packages could be used as well.

AgiMicroRna reads into R [12] the scanned data exported by the `Agilent Feature Extraction` (AFE) image analysis software [1]. Standard graphical utilities can be used to evaluate the quality of the data.

AgiMicroRna includes a full data example that can be loaded into `R` in order to illustrate the capabilities of the package. The data come from human mesenchymal stem cells obtained from bone marrow. 100 ng of each RNA sample were hybridized onto Agilent Human microRNA Microarray v2.0 (G4470B, Agilent Technologies).

The Human microRNA microarray v2.0 contains 723 human and 76 human viral microRNAs, each of them replicated 16 times. There are 362 microRNAs interrogated by 2 different oligonucleotides, 45 microRNAs by 3 and 390 microRNAs interrogated by 4 different oligonucleotides. Only 2 microRNAs are interrogated by the same oligonucleotide. The array contains also a set of positive and negative controls that are replicated a different number of times.

For the statistical analysis we need an estimate of the expression measure for every microRNA that has to be normalized between arrays. This processed signal is going to be used to make statistical inferences about the differential expression. In *AgiMicroRna* the processed microRNA signal can be obtained using two different protocols. The first uses the `Total Gene Signal` (TGS) computed by the AFE algorithm [1] whereas the second obtains an estimate of the gene signal using the `RMA` algorithm [8].

In more detail, the data processing for the first protocol is accomplished according to the following sequential steps: 1) Obtaining the Total microRNA Gene Signal processed by AFE, 2) normalization between arrays. For the RMA algorithm, the steps are slightly different: 1) The signal is background corrected using the exponential + normal convolution model, 2) the background signal

is normalized between arrays, and 3) the total gene signal is estimated from a linear model that takes into account the probe effect. The estimates of the model are obtained using a robust methods such as the median polish.

After obtaining the normalized total gene signal, some of the genes are eliminated from the analysis using some of the quality flags that AFE attaches to each feature. Finally, the processed signal that is going to be used to make statistical inferences is stored in a `ExpressionSet` object [7].

The differential expression analysis is accomplished using the linear model features implemented in *limma* [10]. A linear model is fitted to each microRNA gene so that the fold change between different experimental conditions and their associated standard errors can be estimated. Empirical Bayes methods are applied to obtain moderated statistics [9].

AgiMicroRna contains different functions to extract useful information from the objects generated by *limma*. A list of microRNAs with the different statistics obtained from the differential expression analysis (M value, moderated t and F statistics, p values and FDR, etc) is given. In addition, HTML files that contains links of the declared significant microRNAs to the Sanger miRBase *http://microrna.sanger.ac.uk/* are given. MA plots highlighting the DEGs are also generated.

2 Target File

AgiMicroRna has been primarily designed to produce a processed data to be analyzed using the *limma* package. First, a target file is needed in order to assign each scanned data file to a given experimental group. The target file is a tab-delimited text format file `created by the user` where we specify the factors that are going to be included in the statistical model. The following columns have to be present in the target file. A first column `FileName` is mandatory and includes the image data files names. A second column `Treatment` is also mandatory and includes the treatment effect. The third column, `GErep` is also mandatory, and includes the treatment effect in a numeric code, from 1 to n, being n the number of levels of the treatment effect. Other columns in the target file are optional. They might included information about other explanatory variables specifying other experimental conditions, such age, gender, and blocking variables that take into the account the experimental design (paired, blocked designed, etc). These variables should be included in the target file for its eventual use in the limma model.

In the data example provided in *AgiMicroRna* we use microRNAs that have been measured in human mesenchymal stem cells obtained from bone marrow of 2 independent donors. We want to compare 2 treatments MSC_B and MSC_C with a control MSC_A. For the sake of simplicity we use only 2 replicates for each experimental condition. We define a treatment effect with 3 levels (A,B and C). We need to specify a `GErep` variable to specify the treatment levels using a numeric code, i.e. (1,2,3). In Addition, each treatment has been applied to stem cells that have been obtained from the same individuals, so we have

a randomized blocked (by Subject) design. As we only have two levels of the blocking variable (subject), this is also known as a paired design. To consider the paired design in the statistical analysis we have to add an additional `Subject` variable in the target file that relates the individual to its sample. The target file for this example is shown in Table 1.

FileName	*Treatment*	*GErep*	*Subject*
mscA1.txt	A	1	1
mscA2.txt	A	1	2
mscB1.txt	B	2	1
mscB2.txt	B	2	2
mscC1.txt	C	3	1
mscC2.txt	C	3	2

Table 1: Targe file

After the user have define the target text file specifying their experimental conditions, the target file can be loaded into `R` using the *AgiMicroRna* function `readTargets`.

```
## NOT RUN
> library("AgiMicroRna")
> targets.micro=readTargets(infile="targets.txt",verbose=TRUE)
```

The function `readTargets` returns a `data.frame`. We can use the target included in *AgiMicroRna* to describe the microRNA data used to illustrate the capabilities of the package.

```
> library("AgiMicroRna")
> data(targets.micro)
> print(targets.micro)
```

```
       FileName Treatment GErep Subject
mscA1 mscA1.txt         A     1       1
mscA2 mscA2.txt         A     1       2
mscB1 mscB1.txt         B     2       1
mscB2 mscB2.txt         B     2       2
mscC1 mscC1.txt         C     3       1
mscC2 mscC2.txt         C     3       2
```

3 Reading the data

We have used microRNA data example coming from human mesenchymal stem cells obtained from bone marrow of 2 independent donors. 100 ng of each RNA sample were hybridized onto Agilent Human microRNA Microarray v2.0 (G4470B, Agilent Technologies) The chips were scanned using the Agilent G2567AA

Microarray Scanner System (Agilent Technologies) following manufacturer instructions. Image analysis and data collection were carried out using the `Agilent Feature Extraction 9.1.3.1` (AFE) [1]. with default settings.

To read the scanned data files into `R` we use the `readMicroRnaAFE`. This function creates an object of a class `uRNAList`, similar to the `RGList` object created by *limma* [10], that includes the variables that we need for the data processing and statistical analysis (see Table 2). In particular, the columns "gTotalGeneSignal", "gTotalProbeSignal", "gMeanSignal" and "gProcessedSignal" loaded from the scanned data files, are stored in the following 4 different slots: `TGS`, `TPS`, `meanS` and `procS`. We have created this new class object (adopted from *limma*) to use more appropiate names for the signal values that we use in AgiMicroRna.

The `readMicroRnaAFE` calls a new function `read.agiMicroRna`, similar to the `read.maimages` in *limma*. `read.agiMicroRna` is internally used as follows:

```
## NOT RUN
dd=read.agiMicroRna(targets,
             columns=list(TGS="gTotalGeneSignal",
                          TPS="gTotalProbeSignal",
                          meanS="gMeanSignal",
                          procS="gProcessedSignal"),
                  other.columns=list(IsGeneDetected="gIsGeneDetected",
                                     IsSaturated="gIsSaturated",
                                     IsFeatNonUnifOF="gIsFeatNonUnifOL",
                                     IsFeatPopnOL="gIsFeatPopnOL",
                                     BGKmd="gBGMedianSignal",
                                     BGKus="gBGUsed"),
                    annotation = c( "ControlType", "ProbeName","GeneName"),
                    verbose=TRUE)
```

This implies that in the data files we must have all the columns that are indicated in the calling to `read.agiMicroRna`. If any of those columns are missing, the `readMicroRnaAFE` will produce an error message. In this case, we will have to call the `read.agiMicroRna` by ourselves, omitting those columns that are not present in the data files. For the data pre-processing and differential expression analysis, the columns that we must read at least are: `gTotalGeneSignal`, `gMeanSignal`, `gIsGeneDetected`, `ControlType`, `ProbeName`, and `GeneName`.

A typical use of `readMicroRnaAFE` is like:

```
## NOT RUN
> dd.micro=readMicroRnaAFE(targets.micro,verbose=TRUE)
```

AgiMicroRna contains `uRNAList dd.micro` that can be used to explore the capabilities of the package. `dd.micro` can be loaded into `R` using the `data` command.

```
> data(dd.micro)
> class(dd.micro)
```

```
[1] "uRNAList"
attr(,"package")
[1] ".GlobalEnv"
```

```
> dim(dd.micro)
```

```
[1] 13737     6
```

The variables stored in the `uRNAList` `dd.micro` are shown in Table 2.

```
> print(names(dd.micro))
```

```
[1] "TGS"     "TPS"     "meanS"   "procS"   "targets" "genes"   "other"
```

variable	*data*
dd.micro$TGS	gTotalGeneSignal
dd.micro$TPS	gTotalProbeSignal
dd.micro$meanS	gMeanSignal
dd.micro$procS	gProcessedSignal
dd.micro$targets	File names
dd.micro$genes$ProbeName	Probe Name
dd.micro$genes$GeneName	microRNA Name
dd.micro$genes$ControlType	FLAG to specify the sort of feature
dd.micro$other$gIsGeneDetected	FLAG IsGeneDetected
dd.micro$other$gIsSaturated	FLAG IsSaturated
dd.micro$other$gIsFeatNonUnifOL	FLAG IsFeatNonUnifOL
dd.micro$other$gIsFeatPopnOL	FLAG IsFeatPopnOL
dd.micro$other$gBGMedianSignal	gBGMedianSignal
dd.micro$other$gBGUsed	gBGUsed

Table 2: Variables stored in the `uRNAList` object by `readMicroRnaAFE`

The `ProbeName` is an Agilent-assigned identifier for the probe synthesized on the microarray. The `GeneName` is an identifier for the gene for which the probe provides expression information. The target sequence identified by the systematic name is normally a representative or consensus sequence for the gene.

AFE obtains the `gTotalGeneSignal` as the `TotalProbeSignal` times the number of probes per gene. This signal can be used in the differential expression analysis after a possible normalization step. The `gTotalProbeSignal` is the robust average of all the processed signals for each replicated probe multiplied by the total number of probe replicates. These signals are used by AFE algorithms to estimate the `gTotalGeneSignal`. The `gMeanSignal` is the raw signal for every probe. These signals are processed by AFE to obtain the `gProcessedSignal`. The `gProcessedSignal` is obtained after all the AFE processing steps have been completed. Typically it contains the `Multiplicatively Detrended BackgroundSubtracted Signal` or the `BackgroundSubtractedSignal`. These

signals are used by AFE algorithms to estimate the `gTotalProbeSignal`. The `gBGMedianSignal` is the median raw signal of the local background calculated from intensities of all inlier pixels that represent the local background of the feature. The `gBGUsed` is the background signal computed by AFE algorithms. Usually the `gBGUsed` is the sum of the local background plus the spatial detrending surface value computed by the AFE software. The spatial detrend surface value estimates the noise due to a systematic gradient on the array and it is estimated using the loess algorithm.

AFE attaches to each feature a flag that identifies different quantification errors of the signal. These quantification flags can be used to filter out signals that do not reach a minimum established criterion of quality. `gIsGeneDetected` is a Boolean variable that informs if the gene was detected on the microRNA microarray. This flag considers a probe detected if the signal is three times the error. If one probe of the set of probes comprising a gene is detected, the gene is considered detected,(1 = IsDetected 0 = IsNotDetected). `gIsSaturated` is Boolean flag. 1 indicates that the feature is saturated, i.e. at least half of the inlier pixels in the feature have intensities above the saturation threshold. gIsFeatureNonUnifOL is Boolean flag. 1 indicates that the feature is a non-uniformity outlier; the measured feature pixel variance is greater than the expected feature pixel variance plus the confidence interval. `gIsFeatPopOL` is Boolean flag. 1 indicates that the feature is a population outlier.

Finally, the `dd.micro$targets` contain the name of the files equal to those in target file.

4 Plotting Functions

AgiMicroRna includes functions to create boxplots, density plots, MA plots, `Relative Log Expression` (RLE) [4], and hierarchical cluster of samples that can assist the user in assessing the quality of the data as well as in checking the performance of the processing steps.

All these graphical utilites are included in the `qcPlots` wrapper function. `qcPlots` can be called using different intensity signals. For the `gMeanSignal`, the boxplots, density plots, MA plots, RLE plots and hierachical clustering plots are produced. For the `gProcessedSignal` the same plots are done, except the hierarchical clustering. For the `gTotalProbeSignal` and the `gTotalGeneSignal` only the boxplots and density plots are done, and finally, for the background signals only the boxplots are done.

The MA plots represent the fold-change (M) in the y-axis against the average log expression (A) for two given arrays. To reduce the number of pairwise comparison MA plots displayed, `qcPlots` uses a reference array to which the rest of arrays are compared. The signal values of the reference array are computed as the median spots taken over the whole set of arrays Every kind of feature is identified with different color. of feature is identified with different color.

The RLE plot displays for each sample a Boxplot with the `Relative Log Expression` (RLE) [4]. The RLE is computed for every spot in the array as the

difference between the spot and the median of the same spot across all the arrays. If the majority of the spots are expected not to be differentially expressed, the boxplots should be centered on zero and all of them with approximately the same dispersion.

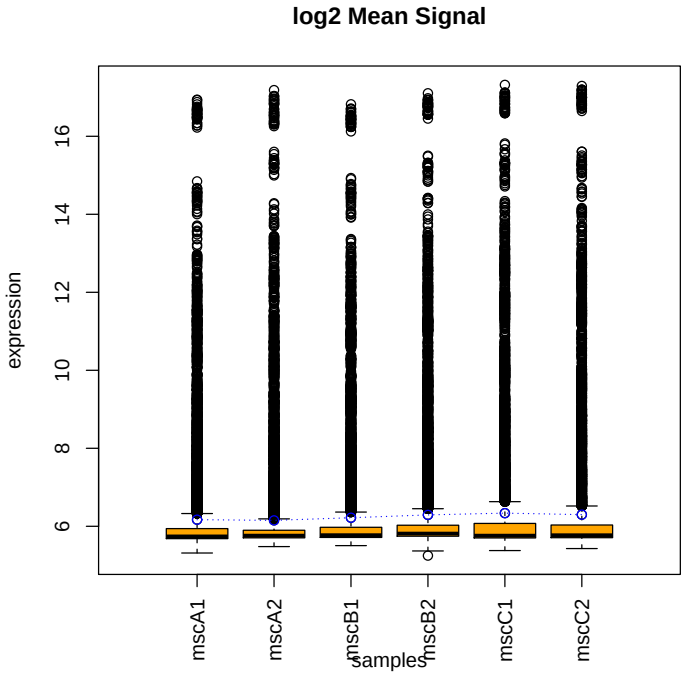
`qcPlots` makes a hierarchical cluster of the samples using the `hclust` function of the *stats* package. We can make a hierarchical clustering of samples either using the whole set of genes or using for instance only the 100 genes that show the highest variability across arrays. The options for the distance measures are `euclidean` and `pearson`. The variables that distinguish the experimental conditions from one another are the differential expressed genes, and that the number of genes may be few relative to the full set of genes of the data set, and hence the cluster analysis will often not reflect the influence of these relevant genes. Therefore if the percentage of differential expression is low, the samples might not be grouped according to their experimental group, since the whole set of genes has very little information regarding the experimental grouping, and the plot will mainly show other grouping features or simply random noise.

A typical call to the `qcPlots` using the Mean Signal intensity is like:

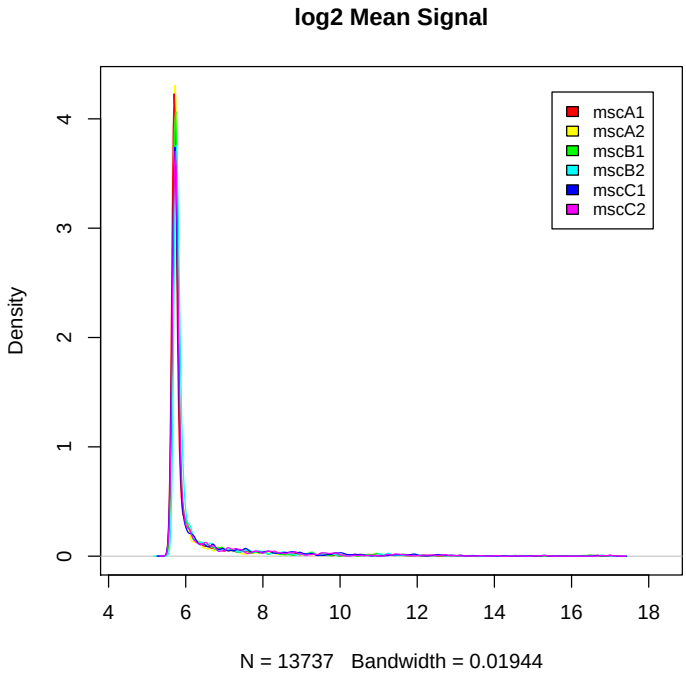
```
> qcPlots(dd.micro,offset=5,
+         MeanSignal=TRUE,
+         ProcessedSignal=FALSE,
+         TotalProbeSignal=FALSE,
+         TotalGeneSignal=FALSE,
+         BGMedianSignal=FALSE,
+         BGUsed=FALSE,
+         targets.micro)
>
```

The same plots can be also generated calling the corresponding functions individually. Next we show how to use these functions using the `gMeanSignal`. Recall, that the `gMeanSignal` is stored in `dd.micro$meanS`.

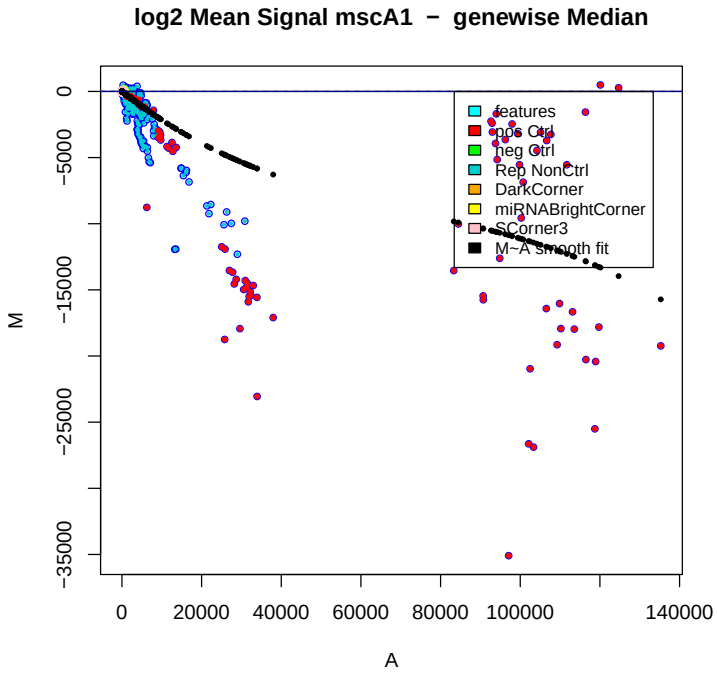
```
> boxplotMicroRna(log2(dd.micro$meanS),
+         maintitle='log2 Mean Signal',
+         colorfill= 'orange')
```


log2 Mean Signal
expression
6
8
10
12
14
16
mscA1
mscA2
mscB1
mscB2
mscC1
mscC2
samples

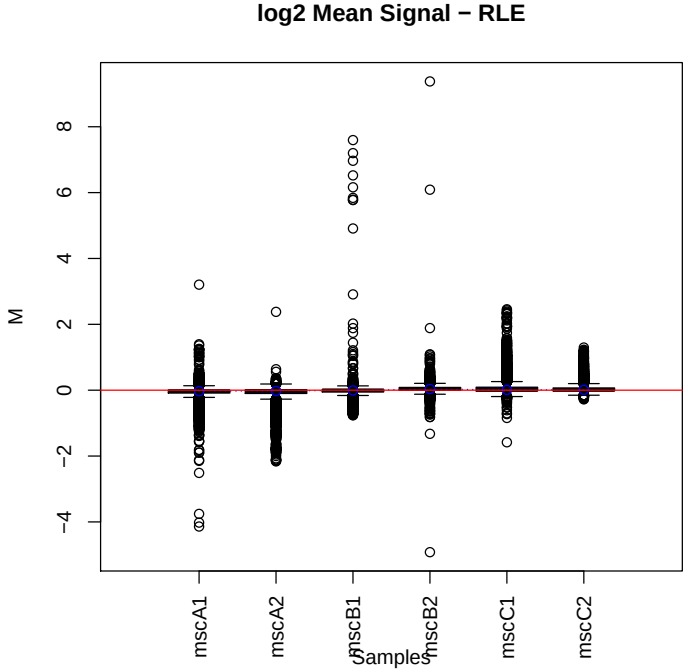
```
> plotDensityMicroRna(log2(dd.micro$meanS),
+          maintitle='log2 Mean Signal')
```



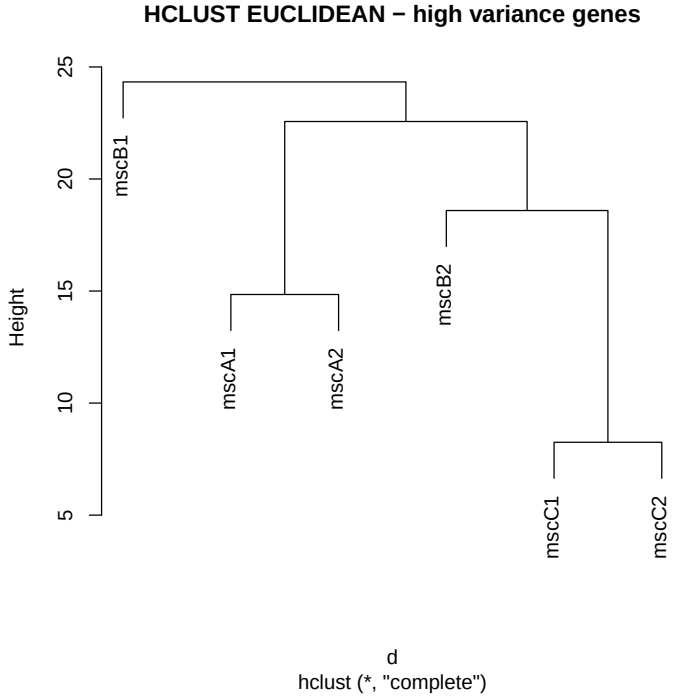
```
> ddaux=dd.micro
> ddaux$G=log2(dd.micro$meanS)
> mvaMicroRna(ddaux,
+         maintitle='log2 Mean Signal',
+         verbose=FALSE)
> rm(ddaux)
>
```



```
> RleMicroRna(log2(dd.micro$meanS),
+          maintitle='log2 Mean Signal - RLE')
```



```
> hierclusMicroRna(log2(dd.micro$meanS),targets.micro$GErep,
+                  methdis="euclidean",
+                  methclu="complete",
+                  sel=TRUE,100)
```



5 Array reproducibility

In the Agilent microRNA platforms normally each microRNA gene is interrogated by 16 probes either using 2 different probes, each of them replicated 8 times, or using 4 different probes replicated 4 times. The probe level replication allows the computation of the coefficient of variation (CV) for each array.

The `cvArray` identifies the replicated non-controlprobes for each feature in the array and computes CV for every microRNA probe set. Then, the median of the CV for each probe set is reported as the array reproducibility. A lower CV median indicates a better array reproducibility.

A set of boxplots shows the CV at a probe level for each array.

To obtain the CV using the `cvArray` function, we can either choose the MeanSignal or ProcessedSignal.

```
> cvArray(dd.micro,
+         "MeanSignal",
```

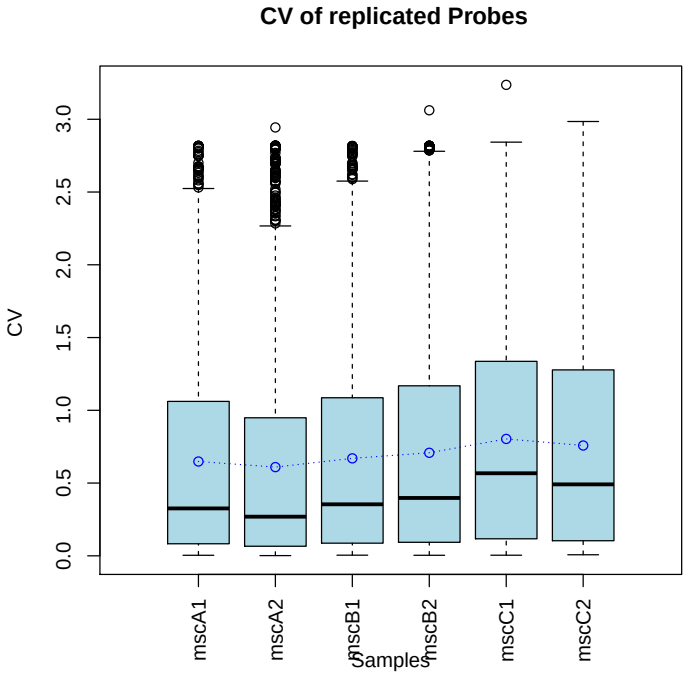
```
+         targets.micro,
+         verbose=TRUE)

Foreground: MeanSignal

        FILTERING BY ControlType FLAG

 RAW DATA:                          13737
 PROBES without CONTROLS:          12784
---------------------------------
  (Non-CTRL) Unique Probe:  2421
  (Non-CTRL) Unique Genes:  799
---------------------------------
DISTRIBUTION OF REPLICATED NonControl Probes
reps
   3    4    5    6    8   16
   1 1558   91   45  724    2
-------------------------------------------------------
Replication at Probe level- MEDIAN  CV
mscA1 mscA2 mscB1 mscB2 mscC1 mscC2
0.326 0.269 0.354 0.398 0.567 0.491
-------------------------------------------------------
DISTRIBUTION OF REPLICATED Noncontrol Genes
reps
 15  16  17
  1 797   1
-------------------------------------------------------

>
```



6 Total Gene Signal

Normally, in the Agilent platfomrs, each microRNA is interrogated by 16 probes either using 2 different probes, each of them replicated 8 times, or using 4 different probes replicated 4 times. In *AgiMicroRna* the processed gene signal that is going to be analyzed can be obtained using two different protocols. The first uses the `gTotalGeneSignal` (TGS) computed by the AFE algorithm [1] and stored in the `uRNAList` (`dd.micro$TGS`) after reading the data. The second option obtains an estimate of the normalized gene signal using the RMA algorithm [8]. The RMA method is explained in the next section.

The function `tgsMicroRna` creates an `uRNAList` containing the `gTotalGeneSignal` processed by the AFE software. This signal might be used for making statistical inferences after a possible normalization step. The TGS processed by AFE [1] may contain negative values. To obtain signals with positive values we can either add a positive small constant (offset) to all the signals or we can select the half option in `tgsMicroRna`, which set to 0.5 all the values smaller than 0.5. To use the `offset` option we have to set `half=FALSE`, otherwise the half method is used by default. The offset option, adds to each signal the quantity `(abs( min(ddTGS$TGS)) + offset)`, where `ddTGS$TGS` is the matrix that contains the `gTotalGeneSignal`.

```
> ddTGS=tgsMicroRna(dd.micro,
+     half=TRUE,
+     makePLOT=FALSE,
+     verbose=FALSE)
>
```

Finally, `tgsMicroRna` stores the estimated TGS in the four fields `ddTGS$TGS` `ddTGS$TPS`, `ddTGS$meanS` and `ddTGS$procS`. Be aware that this TGS is not in log2 scale.

7 Normalization between arrays

To make direct comparisons of data coming from different chips it is important to remove sources of variation of non biological nature that may exists between arrays. Systematic non-biological differences between chips become apparent in several obvious ways especially in labeling and in hybridization, and bias the relative measures on any two chips when we want to quantify the differences in treatment of two samples. Normalization is the attempt to compensate for systematic technical differences between chips, to see more clearly the biological differences between samples.

AgiMicroRna uses two strategies to obtain a gene signal estimate normalized between arrays. The first simply uses the TGS signal processed by the AFE algorithms [1] as it was described in the last section. This TGS can be normalized between arrays using either the `quantile` (default) [3],[5] or the `scale` methods. *AgiMicroRna* incorporates the *limma* function `normalizeBetweenArrays` with options ('none','quantile','scale') [10], [11]. If we use the `none` option the TGS is only log2 transformed.

```
> ddNORM=tgsNormalization(ddTGS,
+                 "quantile",
+                 makePLOTpre=FALSE,
+                 makePLOTpost=FALSE,
+                 targets.micro,
+                 verbose=TRUE)

-------------------------------------------------------
        NORMMALIZATION:         quantile
        OUTPUT in log-2 scale
-------------------------------------------------------
```

`tgsNormalization` creates an `uRNAList` object containing the normalized total gene signal in log 2 scale. If we set makePLOTpre = TRUE and makePLOTpost = TRUE, a density plot, a boxplot, and MA plot, a Relative Log Expression plot (RLE) [4] and a hierarchical clustering plot are constructed using the data before and after normalization, respectively.

The second alternative implemented in *AgiMicroRna* to obtain an estimate of the normalized signal for each microRNA uses the robust multiarray average (RMA) method [8] implemented in the *affy* package [6]. RMA obtains an estimate of the expression measure for each gene using all the probe measures for that gene. First, the signal is background corrected (optional) by fitting a normal + exponential convolution model to the vector of observed intensities. The normal part represents the background and the exponential part represents the signal intensities [8]. Then the arrays are normalized (optional) using the `quantile` normalization [3] ,[5]. Finally, an estimate of the microRNA gene signal is obtained fitting a linear model to , log2 transformed probe intensities. This model produces an estimate of the gene signal taking into account the probe effect. The model parameters estimates are obtained by the median polish algorithm.

The `rmaMicroRna` is a wrapper function that incorporates different function to obtain an RMA estimate for the signal of each microRNA. First, the `rmaMicroRna` the signal can be background corrected using the `rma.background.correct` function of the *preprocessCore* [2], then the signal may be normalized between arrays using the *limma* function `normalizeBetweenArrays` [10], [11], with the `quantile` method [3] ,[5]. Then, the median of the replicated probes is obtained, leading to either 2 or 4 different measures for each gene. These measures correspond to different probes measure for the same gene. These probe measure intensities are log2 transformed and then summarized into a single gene measure by the `rma_c_complete_copy` of the *affy* package [6].

```
> ddTGS.rma=rmaMicroRna(dd.micro,
+                 normalize=TRUE,
+                 background=TRUE)
>
```

Finally, `rmaMicroRna` stores the estimated RMA signal in the four fields `ddTGS.rma$TGS` `ddTGS.rma$TPS`, `ddTGS.rma$meanS` and `ddTGS.rma$procS`. Be aware that this estimated signal estimated by functionrmaMicroRna is now in log2 scale. To get some plots with the gene signal processed by RMA, we can use the code of the following example.

```
> MMM=ddTGS.rma$meanS
> colnames(MMM)=colnames(dd.micro$meanS)
> maintitle='TGS.rma'
> colorfill='blue'
> ddaux=ddTGS.rma
> ddaux$meanS=MMM
> mvaMicroRna(ddaux,maintitle,verbose=TRUE)
> rm(ddaux)
> RleMicroRna(MMM,"RLE TGS.rma",colorfill)
> boxplotMicroRna(MMM,maintitle,colorfill)
> plotDensityMicroRna(MMM,maintitle)
```

In the code above, be aware that `mvaMicroRna` plots by default whatever is contained in `ddaux$meanS`, so first, we create this `ddaux` object and then, we stored in `ddaux$meanS` the matrix we want to use. After the total gene signal have been obtained, either extracting the TGS produced by AFE, or using the RMA algorithm, we can filter out the signals using the FLAGS attached to them by the AFE algorithm.

8 Filtering Probes

The AFE attach to each feature a flag that identifies different quantification errors of the signal that can be used to filter out the microRNAs that do not reach a minimum of quality. This filtering is normally done after the Total Gene Signal has been normalized. Different filtering criteria can be established to be more or less demanding. With low number of replicates probably we need to set more restrictive filtering limits.

The steps that are currently implemented in `filterMicroRna` are to remove control features, to remove gene that are not detected (gIsGeneDetected = 0), and to filter out microRNAs which expression do not reach a certain level based on the expression of the negative controls.

Basically, the gIsGeneDetected filtering removes microRNAs that are not expressed in any experimental condition. To do that, for a given feature xi across samples, we set limit that is the minimum % of features that they must remain in at least one experimental condition with a gIsGeneDetected-FLAG = 1 (Is Detected).

Additionally, if we want to be more demanding, we can remove the microRNAs whose `gMeanSignal` expression are close to the expression of the negative control features. As before se set a limit for the % of microRNAs that they must above a Limit expression value, in at least, one of the experimental conditions. The Limit expression value is defined as Mean(negative control expression) + 1.5 x SD(negative control expression)

The function returns a `uRNAList` containing the FILTERED data. In order to allow the tracking of features that may have been filtered out from the original raw data, the following files are given:

`NOCtrl_FlagIsGeneDetected.txt`: IsGeneDetected Flag for the Non Control Genes, 1 = detected `IsNOTGeneDetected.txt`: Genes that do not are not detected according to IsGeneDetected Flag

To continue the with the illustration we use the total gene signal estimated by the Agilent Feature Extraction software and normalized by `quantile` (`ddNORM`). We could have use the `ddTGS.rma` obtained by RMA as well.

```
> ddPROC=filterMicroRna(ddNORM,
+                       dd.micro,
+                       control=TRUE,
+                       IsGeneDetected=TRUE,
+                       wellaboveNEG=FALSE,
```

```
+                     limIsGeneDetected=75,
+                     limNEG=25,
+                     makePLOT=FALSE,
+                     targets.micro,
+                     verbose=TRUE,
+                     writeout=FALSE)

FILTERING PROBES BY FLAGS


FILTERING BY ControlType

   FEATURES BEFORE FILTERING:  821
         FEATURES AFTER ControlType FILTERING:  799
-----------------------------------------------------
FILTERING BY IsGeneDetected FLAG

        FLAG FILTERING OPTIONS - FLAG OK = 1 - limIsGeneDetected:  75 %
        FEATURES AFTER IsGeneDetected FILTERING:  294
        NON Gene Detected : 505
-----------------------------------------------------
```

9 creating an ExpressionSet object

After all the processing steps the `esetMicroRna` creates an `ExpressionSet` object [7] that can be used for the statistical analysis. If we set makePLOT=TRUE some plots are displayed (Heatmaps, PCAs)

```
> esetPROC=esetMicroRna(ddPROC,
+           targets.micro,
+           makePLOT=FALSE,
+           verbose=TRUE)

outPUT DATA: esetPROC
Features  Samples
     294        6
-----------------------------------------------------
```

The processed data can be written in an output file using the function `writeEset`

```
> writeEset(esetPROC,
+             ddPROC,
+             targets.micro,
+             verbose=TRUE)
```

10 Differential expression analysis using LIMMA

The esetPROC contains the Total Gene Signal for every microRNA that have passed the filtering process, basically, those microRNAs that are expressed in at least one of our experimental conditions. This signals are used to look for the microRNAs that are differentially expressed between our experimental conditions. A linear model is fitted to each microRNA gene so that the fold change between different experimental conditions and their standard errors can be estimated. Empirical Bayes methods are applied to obtain moderated statistics. The functions of the *limma* [10] are used to that end.

10.1 Linear Model

In our case, we had a paired design (by subject) and we want to compare treatment B and C vs. a control treatment A. The linear model that we are going to fit to every microRNA is going to be `y = Treatment + Subject + error term`. This model is going to estimate the treatment effect. Then, the comparison of interest are done by establishing contrasts between the estimates of the treatment effects.

First, we define the factors that we are going to use in our model formula: treatment and subject. We use the structure defined in the `targets.micro` data frame. One way of doing this could be this way:

```
> levels.treatment=levels(factor(targets.micro$Treatment))
>         treatment=factor(as.character(targets.micro$Treatment),
+                 levels=levels.treatment)
>

> levels.subject=levels(factor(targets.micro$Subject))
>         subject=factor(as.character(targets.micro$Subject),
+                 levels=levels.subject)
```

10.2 Fitting the model. Design and Contrast Matrices

To fit the model, we need to define a `design matrix`. The `design matrix` is an incidence matrix that relates each array/sample/file to its given experimental conditions, in our case, relates each file to one of the three treatments and to the subjects (paired design by subject). We can define the `design matrix` using `model.matrix( -1 + treatment + subject)`. This will specify a model without intercept that will give us the estimates for: treatmentA, treatmentB, treatmentC and subject2. `R` use by default the `contr.treatment` parameterization, which means that each level is compared with its baseline level (which is set to 0 to avoid singularities). If we specify a model without intercept, no singularities are produced for the treatment factor, so we get estimates for all its levels. For the subject factor (two levels), we only get an estimate for the subject 2, which is interpreted as the difference with respect to subject1 (baseline = 0).

We can specify the design matrix as:

```
> design=model.matrix(~ -1 + treatment + subject)
> print(design)

  treatmentA treatmentB treatmentC subject2
1          1          0          0        0
2          1          0          0        1
3          0          1          0        0
4          0          1          0        1
5          0          0          1        0
6          0          0          1        1
attr(,"assign")
[1] 1 1 1 2
attr(,"contrasts")
attr(,"contrasts")$treatment
[1] "contr.treatment"

attr(,"contrasts")$subject
[1] "contr.treatment"
```

Then the linear model can be fitted using `lmFit` from *limma* [10]. This will get the treatment estimates for each microRNA in the `ExpressionSet` object.

```
> fit=lmFit(esetPROC,design)
> names(fit)

 [1] "coefficients"     "rank"             "assign"           "qr"
 [5] "df.residual"      "sigma"            "cov.coefficients" "stdev.unscaled"
 [9] "pivot"            "Amean"            "method"           "design"

> print(head(fit$coeff))

               treatmentA treatmentB treatmentC   subject2
hsa-miR-152     7.5721096   7.655674   7.566486 -0.1156653
hsa-miR-15a*    0.9264631   1.065891   1.210531 -0.2241877
hsa-miR-337-5p  6.2447897   7.297695   7.083908 -0.4488954
hsa-miR-129-3p  3.0385412   2.215383   1.110268  0.2156636
hsa-miR-125b   13.0228097  13.022810  13.103000 -0.1069200
hsa-miR-542-5p  3.3726938   2.704901   3.915922 -0.4257320
```

To compare A vs. B and A vs. C, we can define the contrasts of interest using a contrast matrix as in

```
> CM=cbind(MSC_AvsMSC_B=c(1,-1,0,0),
+          MSC_AvsMSC_C=c(1,0,-1,0))
> print(CM)
```

```
     MSC_AvsMSC_B MSC_AvsMSC_C
[1,]            1            1
[2,]           -1            0
[3,]            0           -1
[4,]            0            0
```

And then, we can estimate those contrasts using `contrasts.fit` from *limma* [10].

```
> fit2=contrasts.fit(fit,CM)
> names(fit2)

 [1] "coefficients"     "rank"             "assign"           "qr"
 [5] "df.residual"      "sigma"            "cov.coefficients" "stdev.unscaled"
 [9] "Amean"            "method"           "design"           "contrasts"

> print(head(fit2$coeff))

               MSC_AvsMSC_B MSC_AvsMSC_C
hsa-miR-152     -0.08356481  0.005624054
hsa-miR-15a*    -0.13942829 -0.284067826
hsa-miR-337-5p  -1.05290497 -0.839118503
hsa-miR-129-3p   0.82315829  1.928273625
hsa-miR-125b     0.00000000 -0.080189971
hsa-miR-542-5p   0.66779289 -0.543228542
```

Finally, we can obtain moderated statistics using `eBayes` from *limma* [10], [9].

```
> fit2=eBayes(fit2)
> names(fit2)

 [1] "coefficients"     "rank"             "assign"           "qr"
 [5] "df.residual"      "sigma"            "cov.coefficients" "stdev.unscaled"
 [9] "Amean"            "method"           "design"           "contrasts"
[13] "df.prior"         "s2.prior"         "var.prior"        "proportion"
[17] "s2.post"          "t"                "df.total"         "p.value"
[21] "lods"             "F"                "F.p.value"
```

The function `basicLimma` implemented in *AgiMicroRna* gathers all the last steps in a function to produces the last `MarrayLM` `fit2` object.

This `MarrayLM` object includes, amongst other things, the log fold change of the contrasts (M value stored in `fit2$coeff`), along with its moderated-t statistic (stored in `fit$t`) and its corresponding p value (in `fit2$p.value`). Be aware that to obtain correct inferences these p values must be corrected by some multiple testing procedure.

```
> fit2=basicLimma(esetPROC,design,CM,verbose=TRUE)
```

10.3 Selecting significant microRNAs

`getDecideTests` uses the `decideTests` function from the *limma* package [10] to classify the list of genes as up, down or not significant, taking into account the multiplicity of the tests. `getDecideTests` summarizes the number of up and down genes for every contrasts according to the specified p value threshold. When we have multiple contrasts, the significant genes can be selected using different strategies that are specified by `DEmethod`. In `getDecideTests` only `separate` and `nestedF` options are implemented. See `decideTests` in *limma* [10]

```
> DE=getDecideTests(fit2,
+         DEmethod="separate",
+         MTestmethod="BH",
+         PVcut=0.10,
+         verbose=TRUE)

-------------------------------------------------------
 Method for Selecting DEGs: separate
 Multiple Testing  method:  BH - pval 0.1

     MSC_AvsMSC_B MSC_AvsMSC_C
UP             26           10
DOWN           23           12
-------------------------------------------------------
```

`pvalHistogram` depicts a histogram of the p values. For multiple contrasts, the function creates a histogram for every t.test p value (separate) or a single histogram for the `F.test pvalue` (`nestedF`). The histograms of the pvalues give us an idea about the amount of differential expression that we have in our data. A uniform histogram will indicate no differential expression in the data set, whereas a right skewed histogram, will indicate some significant differential expression

```
> pvalHistogram(fit2,
+         DE,
+         PVcut=0.10,
+         DEmethod="separate",
+         MTestmethod="BH",
+         CM,
+         verbose=TRUE)
```

The function `significantMicroRna` summarizes the results of the differential expression analysis extracting information from the `MArrayLM` and `TestResults` objects generated by the *limma* functions. `significantMicroRna` creates a list containing the genes with their relevant statistics. The significant genes above the `PVcut` p values are also given in a html file that links the selected microRNAs to the `miRBase` *http://microrna.sanger.ac.uk/*. MA plots highlighting the differentially expressed genes are also displayed.

```
> significantMicroRna(esetPROC,
+         ddPROC,
+         targets.micro,
+         fit2,
+         CM,
+         DE,
+         DEmethod="separate",
+         MTestmethod="BH",
+         PVcut=0.10,
+         Mcut=0,
+         verbose=TRUE)
```

When multiple contrasts are done, the method for the selection of the significant genes can be either `separated` or `nestedF`. See `decideTests` in *limma* [10] for a detailed description on these two methods. When `separated` is plugged in into the `significantMicroRna` function then a list including all the genes that have been analyzed is given for each contrast. These lists contain the statistics obtained from the differential expression analysis. The information that is given for each gene is shown in Table 3.

variable	*data*
PROBE	Probe name (one of the probes interrogating the gene)
GENE	microRNA name
M	Fold change
A	Mean of the intensity for that microRNA
t	moderated t-statistic
pval	p value of the t-statistic
adj.pval	p value adjusted by `MTestmethod`
fdr.pval	p value adjusted by fdr

Table 3: Statistics given by `significantMicroRna` for `separate`

The html output files with links to the `miRBase` *http://microrna.sanger.ac.uk/* only includes the microRNAs that have been declared as significant. Sometimes when we correct by multiple testing we cannot declare any single gene as differentially expressed so the html files are not created. Despite of the fact that no gene is been declared significant, the user might be interested in having a link to the `miRBase` for the top ranked differentially expressed genes. When this happens, we can be set `MTestmethod = none`. Since the `adj.pval` is the p value adjusted by the `MTestmethod = none`, then it will be the p value without any correction. in this case, it might be interesting to have the fdr value, despite of the fact that the user has decided not apply any multiple testing correction. That is why, an additional column `fdr.pval` is included containing the p values corrected by fdr, independent of the method that we have specify in `MTestmethod`. Be aware that a multiple testing correction should be used.

If the `nestedF` is used, then two lists are printed out for each contrast. A first set of lists containing the selected significant genes, and a second group of lists containing the rest of the genes that have been analyzed and they do not have been declared significant. The columns given in this case in the output files are shown in Table 4.

variable	*data*
PROBE	Probe name (one of the probes interrogating the gene)
GENE	microRNA name
M	Fold change
A	Mean of the intensity for that microRNA
t	moderated t-statistic
F	F statistic (null hypothesis: Ci = Cj, for all contrasts i, j)
t pval	p value of the t-statistic
adj.F.pval	F p value adjusted by `MTestmethod`
fdr.F.pval	F p value adjusted by fdr

Table 4: Statistics given by `significantMicroRna` for `nestedF`

References

[1] Agilent. *Agilent Feature Extraction Reference Guide*, 2007.

[2] B. M. Bolstad. *preprocessCore: A collection of pre-processing functions*. R package version 1.4.0.

[3] B. M. Bolstad. Probe level quantile normalization of high density oligonucleotide array data. *Unpublished Manuscript*, 2001.

[4] B. M. Bolstad, F. Collin, J. Brettschneider, K. Simpson, L. Cope, R. Irizarry, and T.P. T. P. Speed. *Quality Assesement of Affymetrix GeneChip Data*, pages 397–420. Springer, New York, 2005.

[5] B. M. Bolstad, R. Irizarry, M. Astrand, and T. P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19:185–193, 2003.

[6] Laurent Gautier, Leslie Cope, Benjamin M. Bolstad, and Rafael A. Irizarry. affy—analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004.

[7] R. Gentleman, V.J. Carey, D.M. Bates, B. Bolstad, M Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, et al. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5(43):R80, 2004.

[8] R. Irizarry, B. Hobbs, F. Collin, Y. Beazer-Barclay, K. Antonellis, U. Scherf, and T. P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4:249–264, 2003.

[9] G. K. Smyth. Linear models and empirical bayes methods for assessing diferential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3, 2004.

[10] G. K. Smyth. *Limma: linear models for microarray data*, pages 397–420. Springer, New York, 2005.

[11] G. K. Smyth and T. P. Speed. Normalization of cdna microarray data. *Methods*, 31:265–273, 2003.

[12] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.