Using lumi, a package processing Illumina Microarray

Pan Du[‡,*], Gang Feng[‡,†], Warren A. Kibbe[‡,‡], Simon Lin[‡,§]

May 3, 2016

[‡]Robert H. Lurie Comprehensive Cancer Center
Northwestern University, Chicago, IL, 60611, USA

Contents

1 Overview of lumi

Illumina microarray is becoming a popular microarray platform. The BeadArray technology from Illumina makes its preprocessing and quality control different from other microarray technologies. Unfortunately, until now, most analyses have not taken advantage of the unique properties of the BeadArray system. The *lumi* Bioconductor package especially designed to process the Illumina microarray data, including Illumina Expression and Methylation microarray data. The *lumi* package provides an integrated solution for the bead-level Illumina microarray data analysis. The package covers data input, quality control, variance stabilization, normalization and gene annotation.

For details of processing Illumina methylation microarray, especially Infinium methylation microarray, please check another tutorial in *lumi* package: "Analyze Illumina Infinium methylation microarray data". All following description is focused on processing Illumina expression microarrays.

The *lumi* package provides unique functions for expression microarray processing. It includes a variance-stabilizing transformation (VST) algorithm [2] that takes advantage of the technical replicates available on every Illumina microarray. A robust spline normalization (RSN), which combines the features of the quantile and loess normalization, and simple scaling normalization (SSN) algorithms are also implemented in this package. Options available in other popular normalization methods are also provided. Multiple quality control plots for expression and control probe data are provided in the package. To better annotate the Illumina data, a new, vendor independent nucleotide universal identifier (nuID) [3] was devised to identify the probes of Illumina microarray. The nuID

[*]dupan.mail (at) gmail.com
[†]g-feng (at) northwestern.edu
[‡]wakibbe (at) northwestern.edu
[§]s-lin2 (at) northwestern.edu

indexed Illumina annotation packages is compatible with other Bioconductor annotation packages. Mappings from Illumina Target Id or Probe Id to nuID are also included in the annotation packages. The output of lumi processed results can be easily integrated with other microarray data analysis, like differentially expressed gene identification, gene ontology analysis or clustering analysis.

2 Citation

For the people using lumi package, please cite the following papers in your publications.

* For the package:

Du, P., Kibbe, W.A. and Lin, S.M., (2008) 'lumi: a pipeline for processing Illumina microarray', Bioinformatics 24(13):1547-1548

* For the VST (variance stabilization transformation) algorithm, please cite:

Lin, S.M., Du, P., Kibbe, W.A., (2008) 'Model-based Variance-stabilizing Transformation for Illumina Microarray Data', Nucleic Acids Res. 36, e11

* For nuID annotation packages, please cite:

Du, P., Kibbe, W.A. and Lin, S.M., (2007) 'nuID: A universal naming schema of oligonucleotides for Illumina, Affymetrix, and other microarrays', Biology Direct, 2, 16.

Thanks for your help!

3 Installation of lumi package

In order to install the lumi package, the user needs to first install R, some related Bioconductor packages. You can easily install them by the following codes.

```
source("http://bioconductor.org/biocLite.R")
biocLite("lumi")
```

For the users want to install the latest developing version of lumi, which can be downloaded from the developing section of Bioconductor website. Some additional packages may be required to be installed because of the update the Bioconductor. These packages can also be found from the developing section of Bioconductor website. You can also directly install the source packages from the Bioconductor website by specify the developing version number, which can be found at the Bioconductor website. Suppose the developing version is 2.3, to install the latest lumi pakcage in the Bioconductor developing version, you can use the following command:

```
## replace "xxx" with the Bioconductor version number.
install.packages("lumi",repos="http://www.bioconductor.org/packages/xxx/bioc",type="source
```

An Illumina benchmark data package *lumiBarnes* can be downloaded from Bioconductor Experiment data website.

4 Object models of major classes

The *lumi* package has one major class: **LumiBatch**. **LumiBatch** is inherited from **ExpressionSet** class in Bioconductor for better compatibility. Their relations are shown in Figure **??**. **LumiBatch** class includes *se.exprs*, *beadNum* and *detection* in **assayData** slot for additional informations unique to Illumina microarrays. A controlData slot is used to keep the control probe information, and a QC slot is added for keeping the quality control information. The S4 function `plot` supports different kinds of plots by specifying the specific plot type of **LumiBatch** object. See help of `plot-methods` function for details. The *history* slot records all the operations made on the **LumiBatch** object. This provides data provenance. Function `getHistory` is to retrieve the *history* slot. Please see the help files of **LumiBatch** class for more details. A series of functions: `lumiR`, `lumiR.batch`, `lumiB`, `lumiT`, `lumiN` and `lumiQ` were designed for data input, preprocessing and quality control. Function `lumiExpresso` encapsulates the preprocessing methods for easier usability.

5 Data preprocessing

The first thing is to load the *lumi* package.

```
> library(lumi)
```

5.1 Intelligently read the BeadStudio output file

The `lumiR` function supports directly reading the Illumina raw data output of the Illumina Bead Studio toolkit from version 1 to version 3. It can automatically detect the BeadStudio output version and format and create a new **LumiBatch** object for it. An example of the input data format is shown in in Figure **??**. For simplicity, only part of the data of first sample is shown. The data in the highlighted columns are kept in the corresponding slots of **LumiBatch** object, as shown in Figure **??**. The `lumiR` function will automatically determine the starting line of the data. The columns with header including `AVG_Signal` and `BEAD_STD` are required for the **LumiBatch** object. By default, the sample IDs and sample labels are extracted from the column names of the data file. For example, based on the column name: `AVG_Signal-1304401001_A`, we will extract `"1304401001"` as the sample ID and `"A"` as the sample label (The function assumes the separation of the sample ID and the sample label is `"_"` if it exists in the column name.). The function will check the uniqueness of sample IDs. If the sample ID is not unique, the entire portion after removing `"AVG_Signal"` will be used as a sample ID. The user can suppress this parsing by setting the parameter "parseColumnName" as FALSE.

The `lumiR` will automatically initialize the QC slot of the **LumiBatch** object by calling `lumiQ`. If BeadStudio outputted the control probe data, their information will be kept in the controlData slot of the **LumiBatch** object. If BeadStudio outputted the sample summary information, which is called [Samples Table] in the output text file, the information will be kept in BeadStudioSummay within the QC slot of the **LumiBatch** object.

The BeadStudio can output the gene profile or the probe profile. As the probe profile provides unique mapping from the probe Id to the expression pro-

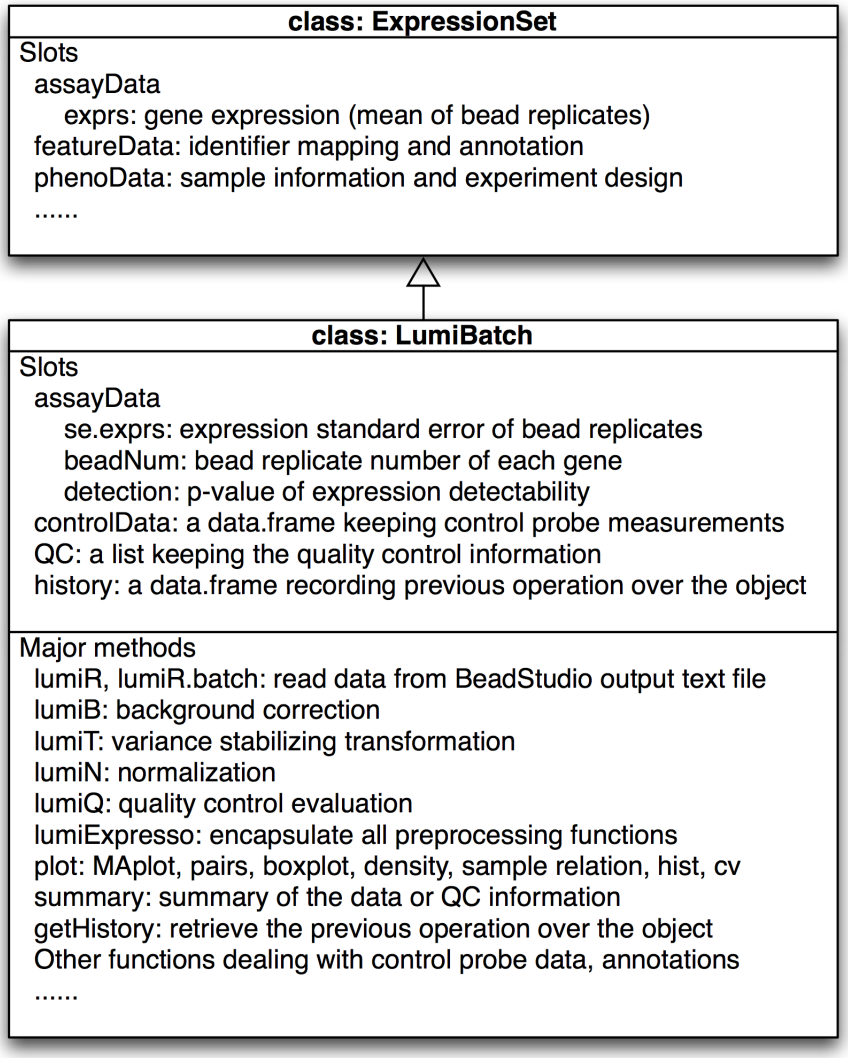


Figure 1: Object models in lumi package

file, outputting probe profile is preferred. When the probe profile is outputted, as show in Figure **??**(B), the ProbeId column will be used as the identifier of **LumiBatch** object.

We strongly suggest outputting the header information when using BeadStudio, as shown in Figure **??**. Please refer to the separate document ("Resolve the Inconsistency of Illumina Identifiers through nuID Annotation") in the lumi package for more details of the changing of BeadStudio output formats.

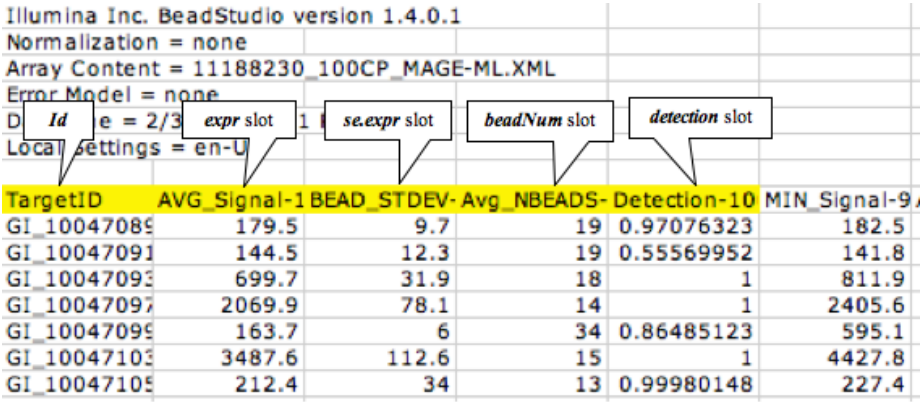
The recent version of BeadStudio can also output the annotation information together with the expression data. In the users also want to input the annotation information, they can set the parameter "inputAnnotation" as TRUE. At the same time, they can also specify which columns to be inputted by setting parameter "annotationColumn". The BeadStudio annotation columns include: SPECIES, TRANSCRIPT, ILMN_GENE, UNIGENE_ID, GI, ACCESSION, SYMBOL, PROBE_ID, ARRAY_ADDRESS_ID, PROBE_TYPE, PROBE_START, PROBE_SEQUENCE, CHROMOSOME, PROBE_CHR_ORIENTATION, PROBE_COORDINATES, DEFINITION, ONTOLOGY_COMPONENT, ONTOLOGY_PROCESS, ONTOLOGY_FUNCTION, SYNONYMS, OBSOLETE_PROBE_ID. As the annotation data is huge, by default, we only input: ACCESSION, SYMBOL, PROBE_START, CHROMOSOME, PROBE_CHR_ORIENTATION, PROBE_COORDINATES, DEFINITION. As some annotation information may be outdated. We recommend using Bioconductor annotation packages to retrieve the annotation information.

For convenience, another function `lumiR.batch` is designed to input files in batch. Basically it combines the output of each file. See the help of `lumiR.batch` for details. `lumiR.batch` function also allows users to add sample information in the phenoData slot of the LumiBatch object. This will be useful in the data analysis. The sampleInfoFile parameter is optional. The file is a Tab-separated text file. The first ID column is required. It represents sample ID, which is defined based on the column names of BeadStudio output file. For example, sample ID of column "1881436070_A_STA.AVG_Signal" is "1881436070_A_STA". The sample ID column can also be found in the "Samples Table.txt" file output by BeadStudio. Another "Label" column (if provided) will be used as the sampleNames of LumiBatch object. All information of sampleInfoFile will be directly added in the phenoData slot in LumiBatch object, which can be retrieved by the command like: pData(phenoData(x.lumi)).

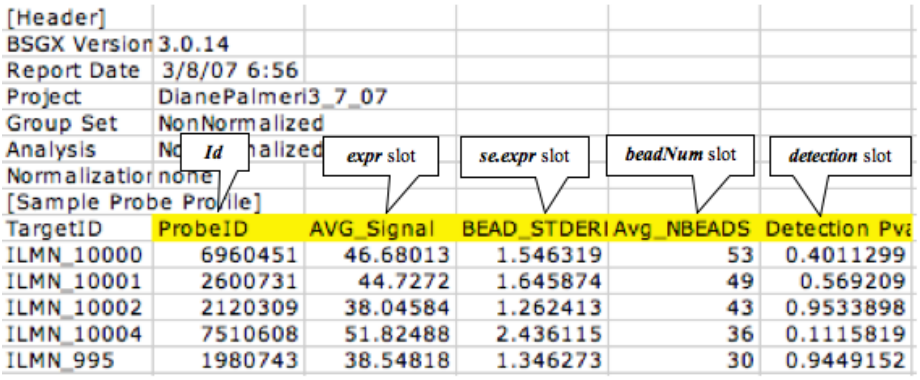
```
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'
> ## load the data
> # x.lumi <- lumiR.batch(fileName, sampleInfoFile='sampleInfo.txt')                # Not
```

Here, we just load the pre-saved example data, example.lumi, which is a subset of the experiment data package *lumiBarnes* in the Bioconductor. The example data includes four samples "A01", "A02", "B01" and "B02". "A" and "B" represent different Illumina slides (8 microarrays on each slide), and "01" and "02" represent different samples. That means "A01" and "B01" are technique replicates at different slides, the same for "A02" and "B02".

```
> ## load example data (a LumiBatch object)
> data(example.lumi)
```



(A) BeadStudio version 1



(B) BeadStudio version 3

Figure 2: An example of the input data format

```
> ## summary of the example data
> example.lumi

Summary of data information:
Illumina Inc. BeadStudio version 1.4.0.1
                Normalization = none
                Array Content = 11188230_100CP_MAGE-ML.XML
                Error Model = none
                DateTime = 2/3/2005 3:21 PM
                Local Settings = en-US


Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
                                              command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                              lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6

Object Information:
LumiBatch (storageMode: lockedEnvironment)
assayData: 8000 features, 4 samples
  element names: beadNum, detection, exprs, se.exprs
protocolData: none
phenoData
  sampleNames: A01 A02 B01 B02
  varLabels: sampleID label
  varMetadata: labelDescription
featureData
  featureNames: oZsQEQXp9ccVIlwoQo 9qedFRd_5Cul.ueZeQ ...
    33KnLHy.RFaieogAF4 (8000 total)
  fvarLabels: TargetID
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: lumiHumanAll.db
Control Data: Available
QC information: Please run summary(x, 'QC') for details!
```

5.2 Quality control of the raw data

The quality control of a **LumiBatch** object includes a data summary (the mean and standard deviation, sample correlation, detectable probe ratio of each sample (microarray)), different quality control plots, and the control probe information.

BeadStudio will usually separately output (or attached after the expressed

data in the same file) the control probe (gene) information, usually named as "Control Probe Profile.txt". The controlData slot in LumiBatch class is designed to keep the control probe (gene) information. The control probe file can be inputted by using function `getControlData` or directly add it to a LumiBatch object by using function `addControlData2lumi`. Several functions `plotControlData`, `plotHousekeepingGene` and `plotStringencyGene` are designed to plot control probe data. Please see their help files for more details.

`LumiQ` function will produce the data summary of a **LumiBatch** object and organize the results in a QC slot of **LumiBatch** object. When creating the **LumiBatch** object, the `LumiQ` function will be called to initialize the QC slot of the **LumiBatch** object.

Summary of the quality control information of example.lumi data. If the QC slot of the **LumiBatch** object is empty, function `lumiQ` will be automatically called to estimate the quality control information.

```
> ## summary of the quality control
> summary(example.lumi, 'QC')

Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.3240  8.568  8.2580  8.3470
standard deviation       1.5580  1.686  1.7230  1.6690
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 76.9500 65.280 88.3200 49.1100

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
                                    command lumiVersion
1 lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                   lumiQ(x.lumi = x.lumi)       1.1.6
```

The S4 method `plot` can produce the quality control plots of **LumiBatch** object. The quality control plots includes: the density plot (Figure **??**), box plot (Figure **??**), pairwise scatter plot between microarrays (Figure **??**) or pair scatter plot with smoothing (Figure **??**), pairwise MAplot between microarrays (Figure **??**) or MAplot with smoothing (Figure **??**), density plot of coefficient of varience, (Figure **??**), and the sample relations (Figure **??**). More details are in the help of `plot,LumiBatch-method` function. Most of these plots can also be plotted by the extended general functions: `density` (for density plot), `boxplot`, `MAplot`, `pairs` and `plotSampleRelation`.

Figure **??** shows the density plot of the **LumiBatch** object by using `plot` or `density` functions.

```
> ## plot the density
> plot(example.lumi, what='density')
> ## or
> density(example.lumi)
```

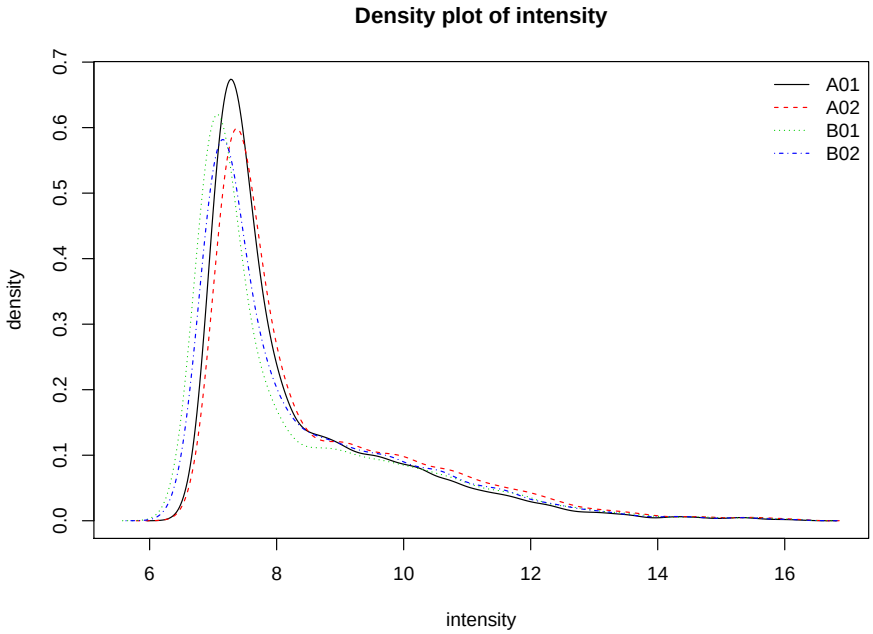


Figure 3: Density plot of Illumina microarrays before normalization

Figure **??** shows the density plot of the **LumiBatch** object by using `plot` or `density` functions.

```
> ## plot the density
> plot(example.lumi, what='density')
> ## or
> density(example.lumi)
```

Figure **??** shows the plot of Cumulative Distribution Function (CDF) from high to low value or in reverse of a **LumiBatch** object by using `plotCDF` function. Comparing with the density plot, the CDF plot in reverse direction can better show the different at the high and middle expression range among different samples.

```
> ## plot the CDF plot
> plotCDF(example.lumi, reverse=TRUE)
```

Figure **??** shows the pairwise sample correlation of the **LumiBatch** object by using `plot` or `pairs` functions.

```
> ## plot the pair plot
> plot(example.lumi, what='pair')
> ## or
> pairs(example.lumi)
> ## pairwise scatter plot with smoothing
> pairs(example.lumi, smoothScatter=T)
```

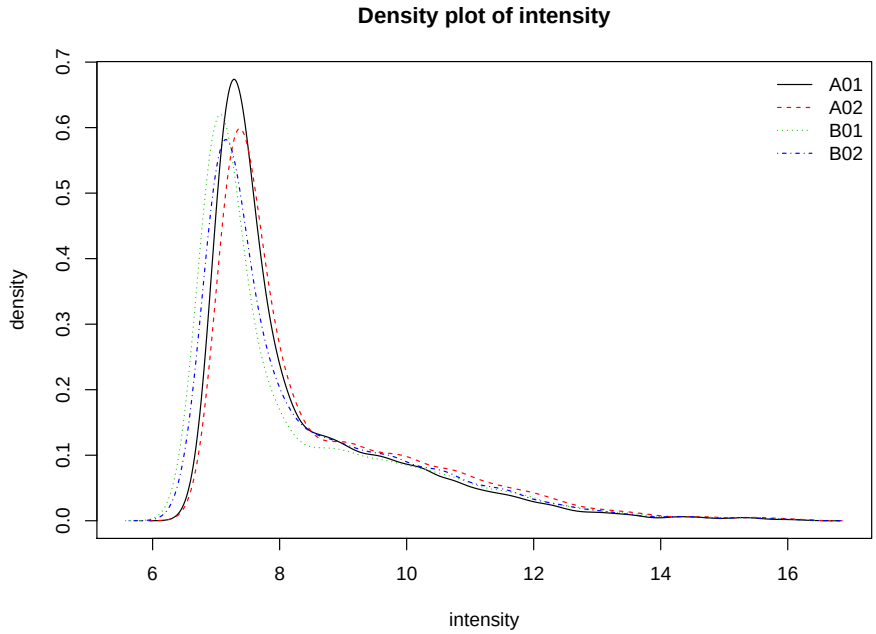


Figure 4: Density plot of Illumina microarrays before normalization

Figure **??** shows the MA plot of the **LumiBatch** object by using `plot` or `MAplot` functions.

```
> ## plot the MAplot
> plot(example.lumi, what='MAplot')
> ## or
> MAplot(example.lumi)
> ## with smoothing
> MAplot(example.lumi, smoothScatter=TRUE)
```

The density plot of the coefficient of variance of the **LumiBatch** object. See Figure **??**. Figure **??** shows the density plot of the coefficient of variance of the **LumiBatch** object by using `plot` function.

Figure **??** shows the sample relations using hierarchical clustering.

Figure **??** shows the sampleRelation using MDS. The color of the sample is based on the sample type, which is `"01", "02", "01", "02"` for the sample data. Please see the help of `plotSampleRelation` and `plot-methods` for more details.

```
> ## plot the sample relations
> plot(example.lumi, what='sampleRelation', method='mds', color=c("01", "02", "01", "02"))
> ## or
> plotSampleRelation(example.lumi, method='mds', color=c("01", "02", "01", "02"))
```

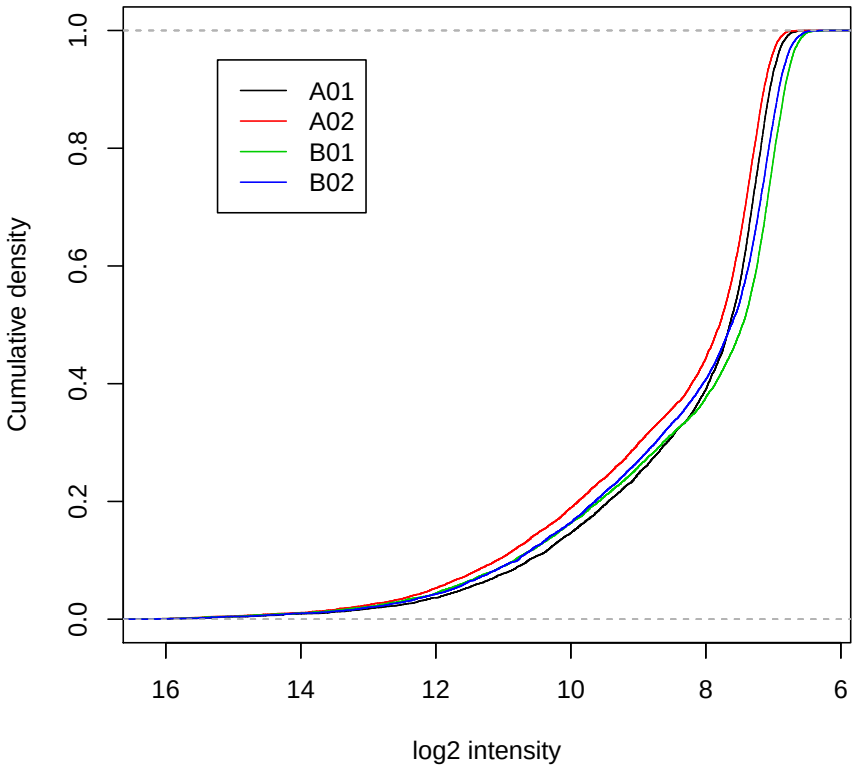


Figure 5: Cumulative Distribution Function (CDF) plot of Illumina microarrays before normalization

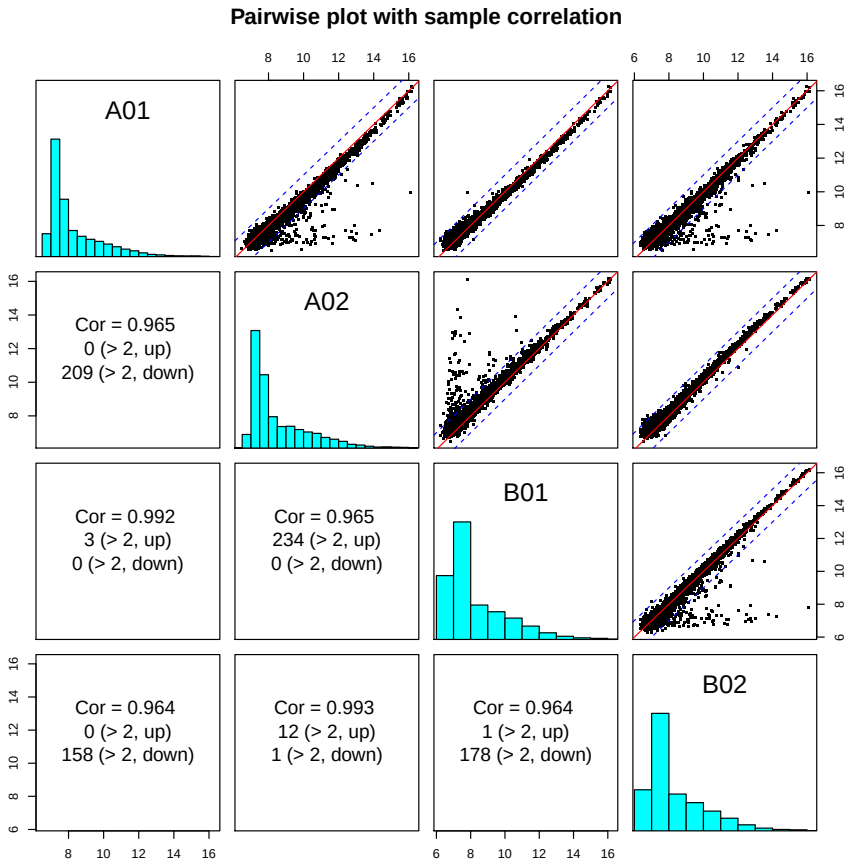


Figure 6: Pairwise plot with microarray correlation before normalization

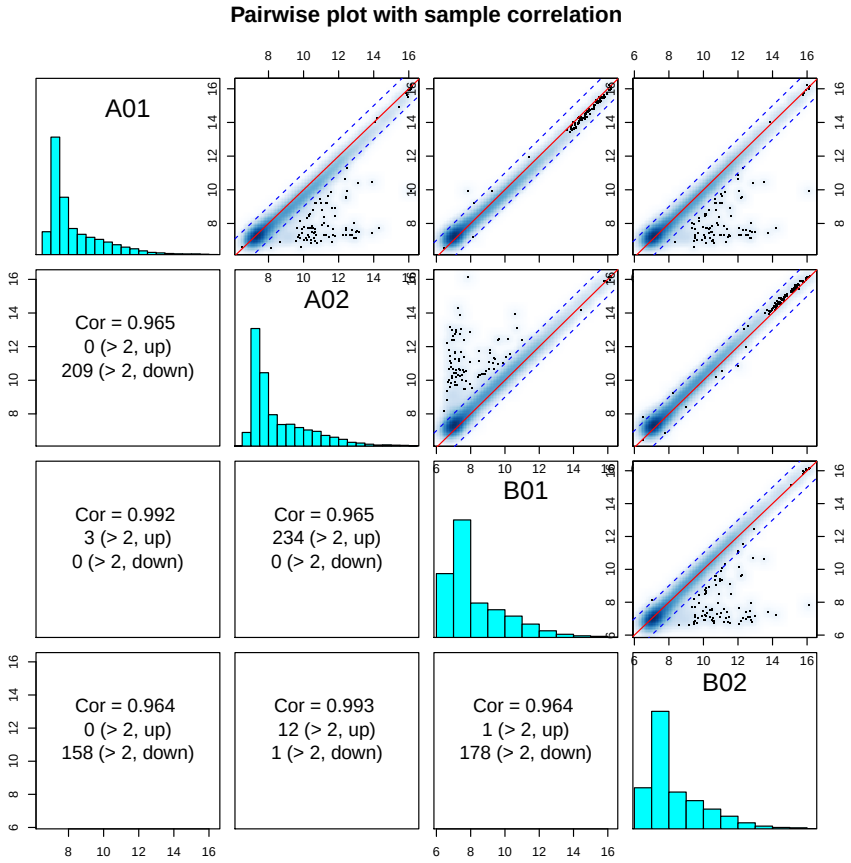


Figure 7: Pairwise plot with microarray correlation before normalization

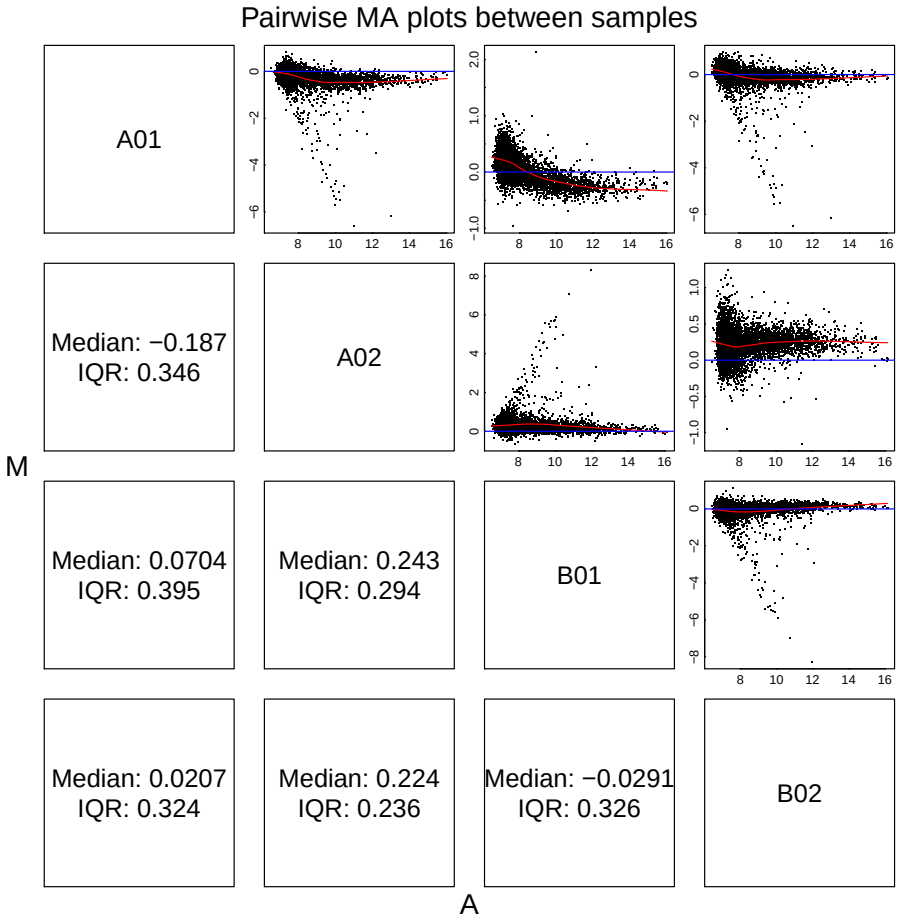


Figure 8: Pairwise MAplot before normalization

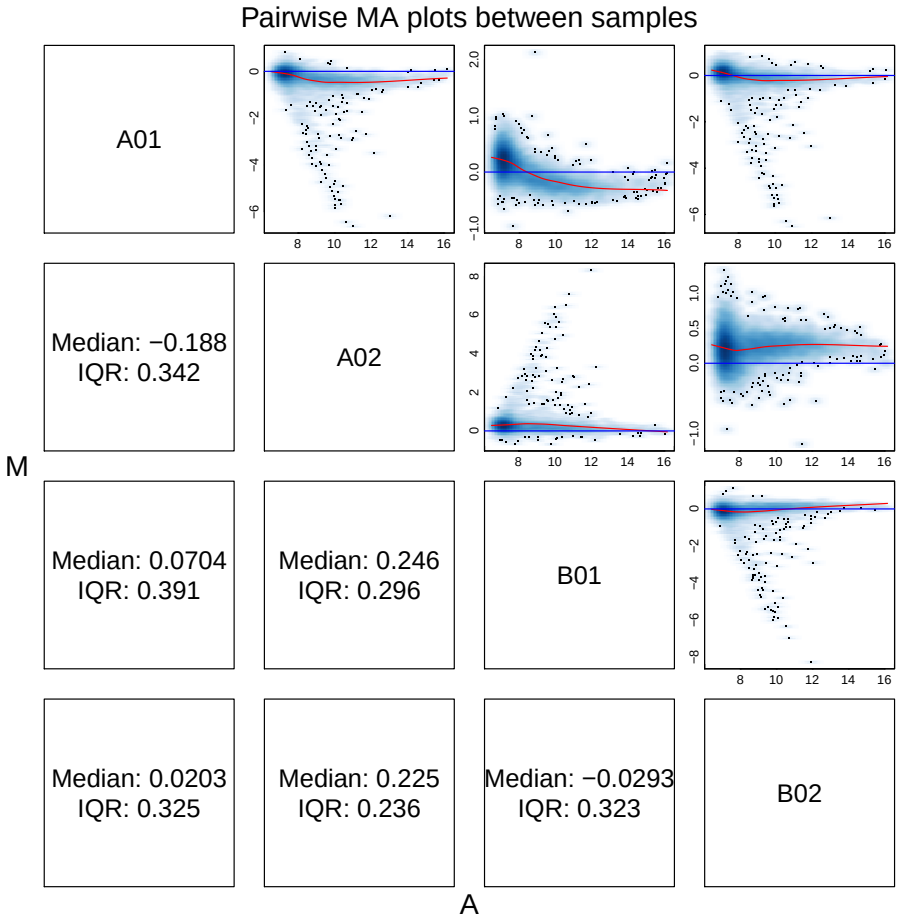


Figure 9: Pairwise MAplot with smoothing before normalization

```
> ## density plot of coefficient of varience
> plot(example.lumi, what='cv')
```

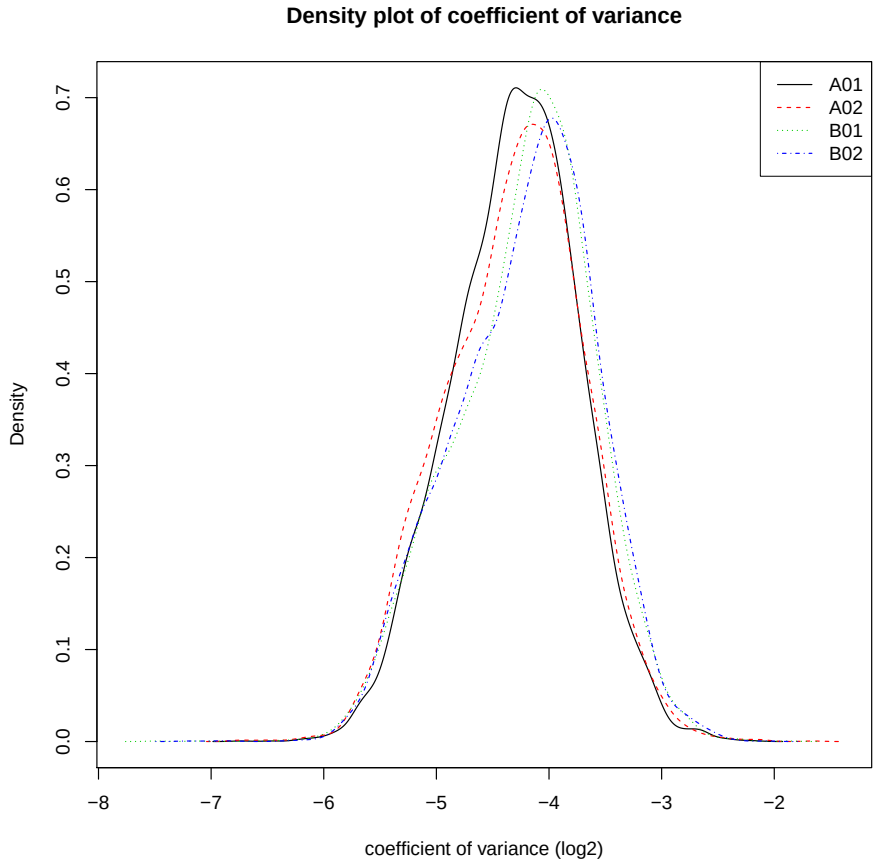


Figure 10: Density Plot of Coefficient of Varience

```
> plot(example.lumi, what='sampleRelation')
```

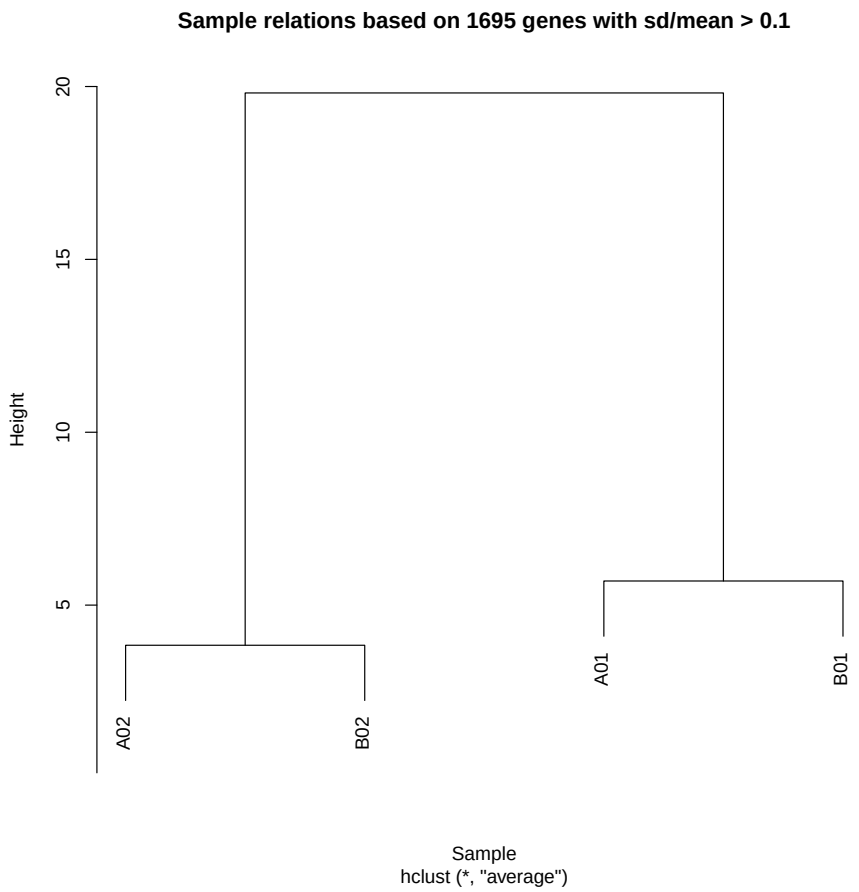


Figure 11: Sample relations before normalization

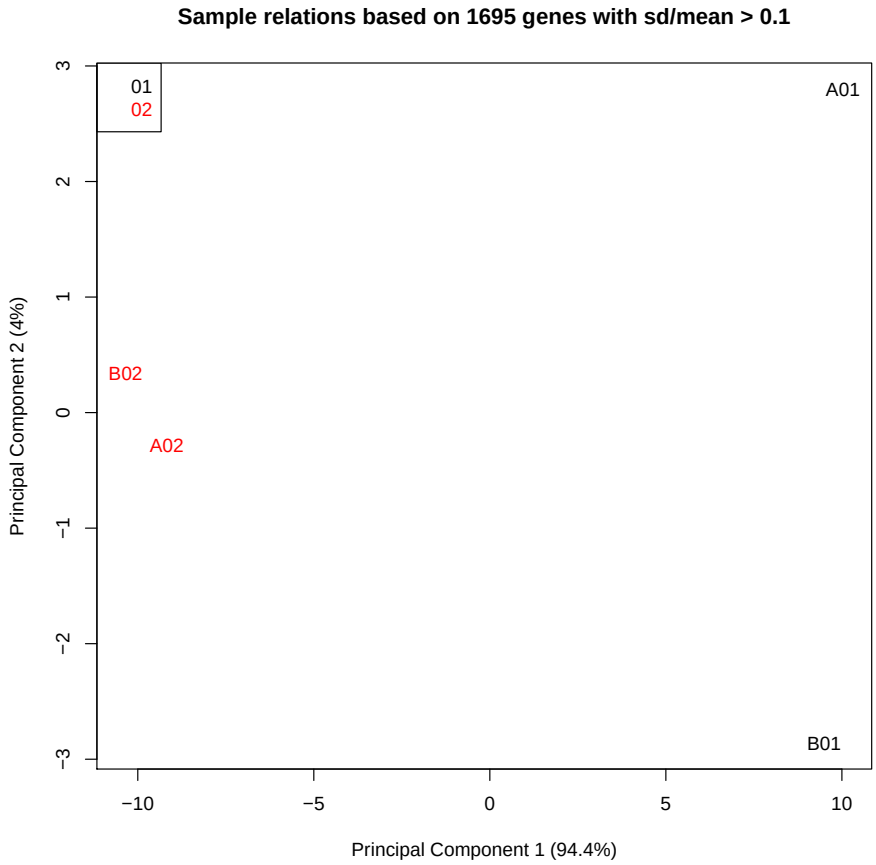


Figure 12: Sample relations before normalization

5.3 Background correction

The *lumi* package provides `lumiB` function for background correction. We suppose the BeadStudio output data has been background corrected. Therefore, no background correction used by default. A method 'bgAdjust' is designed to approximate what BeadStudio does for background adjustment. In the case when 'log2' transform is used in the `lumiT` step, the background correction method ('forcePositive') will be automatically used, which basically adds an offset (minus minimum value plus one) if there is any negative values to force all expression values to be positive. If users are more interested in the low level background correction, please refer to the package *beadarray* for more details. Users can also provide their own background correction function with a LumiBatch Object as the first argument and return a LumiBatch Object with background corrected. See `lumiB` help document for more details.

5.4 Variance stabilizing transform

Variance stabilization is critical for subsequent statistical inference to identify differential genes from microarray data. We devised a variance-stabilizing transformation (VST) by taking advantages of larger number of technical replicates available on the Illumina microarray. Please see [2] for details of the algorithm.

Because the STDEV (or STDERR) columns of the BeadStudio output file is the standard error of the mean of the bead intensities corresponding to the same probe. (Thanks Gordon Smyth kindly provided this information!). As the variance stabilization (see help of `vst` function) requires the information of the standard deviation instead of the standard error of the mean, the value correction is required. The corrected value will be x * sqrt(N), where x is the old value (standard error of the mean), N is the number of beads corresponding to the probe. The parameter 'stdCorrection' of `lumiT` determines whether to do this conversion and is effective only when the 'vst' method is selected. By default, the parameter 'stdCorrection' is TRUE.

Function `lumiT` performs variance stabilizing transform with both input and output being **LumiBatch** object.

Do default VST variance stabilizing transform

```
> ## Do default VST variance stabilizing transform
> lumi.T <- lumiT(example.lumi)

Perform vst transformation ...
2016-05-03 18:57:56 , processing array  1
2016-05-03 18:57:56 , processing array  2
2016-05-03 18:57:56 , processing array  3
2016-05-03 18:57:56 , processing array  4
```

The `plotVST` can plot the transformation function of VST, see Figure **??**, which is close to log2 at high expression values, see Figure **??**. Function `lumiT` also provides options to do `"log2" or "cubicRoot"` transform. See help of `lumiT` for details.

```
> ## plot VST transformation
> trans <- plotVST(lumi.T)
```

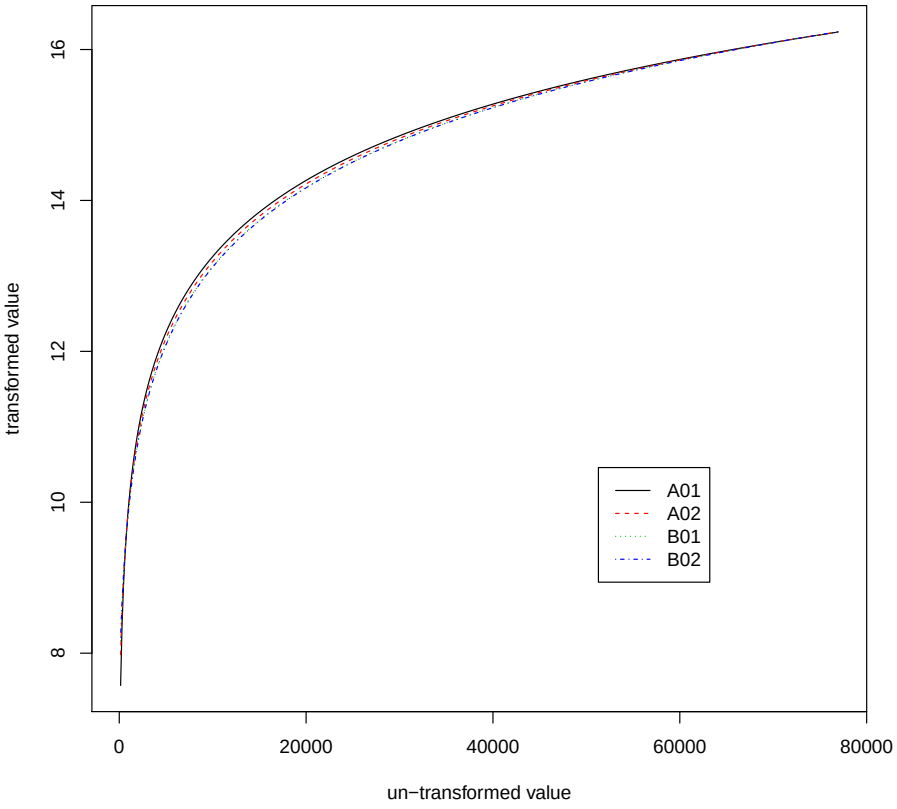


Figure 13: VST transformation

```
> ## compare the log2 and VST transform
> matplot(log2(trans$untransformed), trans$transformed, main='compare VST and log2 transfo
```

5.5 Data normalization

lumi package provides several normalization method options, which include quantile, SSN (Simple Scaling Normalization), RSN (Robust Spline Normalization), loess normalization and Rank Invariant Normalization.

Comparing with other normalization methods, like quantile and curve-fitting methods, SSN is a more conservative method. The only assumption is that each sample has the same background level and the same scale (if do scaling). It basically make all the samples have the same background level and the same scale comparing to the background (if do scaling). There are three methods ('density', 'mean' and 'median') for background estimation. If bgMethod is 'none', then no background adjustment. For the 'density' bgMethod, it estimates the background based on the mode of probe intensities based on the assumption that the background level intensity is the most frequent value across all the probes

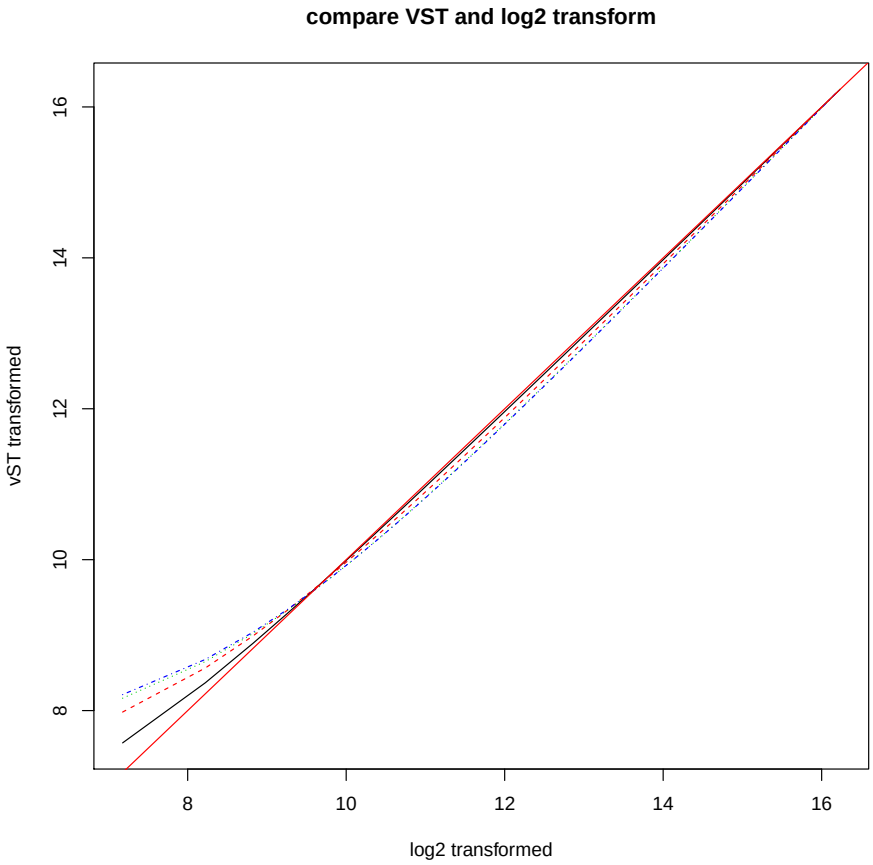


Figure 14: Compare VST and log2 transform

in the chip. For the foreground level estimation, it also provides three methods ('mean', 'density', 'median'). For the 'density' fgMethod, it assumes the background probe levels are symmetrically distributed. The foreground levels were estimated by taking the intensity mean of all other probes except from the background probes. For the 'mean' and 'median' methods (for both bgMethod and fgMethod), it basically estimates the level based on the mean or median of all probes of the sample. If the fgMethod is the same as bgMethod (except 'density' method), no scaling will be performed.

Another normalization method which is unique in the *lumi* package is the Robust Spline Normalization (RSN) algorithm. RSN combines the features of quanitle and loess nor-malization. The advantages of quantile normalization include computational efficiency and preserving the rank order of genes. However, the intensity transformation of a quantile normalization is discontinuous because the normalization forces the intensity values for different samples (microarrrays) having exactly the same distribution. This can cause small differences among intensity values to be lost. In contrast, the loess or spline normalization provides a continuous transformation. However, these methods cannot ensure that the rank of the probes remain unchanged across samples. Moreover, the loess normalization assumes the majority of the genes measured by the probes are non-differentially expressed and their distribution is approximately symmetric, which may not be a good assumption. To address some of these concerns, we developed a Robust Spline Normalization (RSN) method, which combines features from loess and quantile normalization methods. We use a monotonic spline to calibrate one microarray to the reference microarray. To increase the robustness of the spline method, we down-weight the contributions of probes of putatively differentially expressed genes. The probe intensities that are from potentially differentially expressed genes are heuristically determined as follows: First, we run a quantile normalization. Next, we estimate the fold-change of a gene measured by a probe based on the quantile-normalized data. The weighting factor for a probe is calculated based on a Gaussian window function. More details will be shown in a separate manuscript.

The default normalization method used in the Illumina BeadStudio software is Rank Invariant Normalization. In order to support similar functionalities, the *lumi* package also provides a similar normalization implementation call "rankinvariant" (We thanks Arno Velds implemented this function.). Please check the help of `rankinvariant` for more details.

By default, function `lumiN` performs popular quantile normalization. `lumiN` also provides other options to do `"rsn", "ssn", "loess", "vsn", "rankinvariant"` normalization. See help of `lumiN` for details.

Do default quantile between microarray normaliazation

```
> ## Do quantile between microarray normaliazation
> lumi.N <- lumiN(lumi.T)

Perform quantile normalization ...
```

Users can also easily select other normalization method. For example, the following command will run RSN normalization.

```
> ## Do RSN between microarray normaliazation
> lumi.N <- lumiN(lumi.T, method='rsn')
```

5.6 Quality control after normalization

To make sure the data quality meets our requirement, we do a second round of quality control of normalized data with different QC plots. Compare the plots before and after normalization, we can clearly see the improvements.

```
> ## Do quality control estimation after normalization
> lumi.N.Q <- lumiQ(lumi.N)

Perform Quality Control assessment of the LumiBatch object ...

> ## summary of the quality control
> summary(lumi.N.Q, 'QC')                ## summary of QC

Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.8430  8.843  8.8430  8.8430
standard deviation       1.3350  1.335  1.3350  1.3350
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 15.3300 15.080 15.3200 15.4500

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
5 2016-05-03 18:57:56 2016-05-03 18:57:56
6 2016-05-03 18:57:56 2016-05-03 18:57:56
7 2016-05-03 18:57:56 2016-05-03 18:57:56
                                          command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                             lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6
5                        lumiT(x.lumi = example.lumi)      2.24.0
6                              lumiN(x.lumi = lumi.T)      2.24.0
7                              lumiQ(x.lumi = lumi.N)      2.24.0
```

5.7 Encapsulate the processing steps

The `lumiExpresso` function is to encapsulate the major functions of Illumina preprocessing. It is organized in a similar way as the `expresso` function in *affy* package. The following code basically did the same processing as the previous multi-steps and produced the same results lumi.N.Q.

```
> ## Do all the default preprocessing in one step
> lumi.N.Q <- lumiExpresso(example.lumi)
```

```
> plot(lumi.N.Q, what='density')                    ## plot the density
```

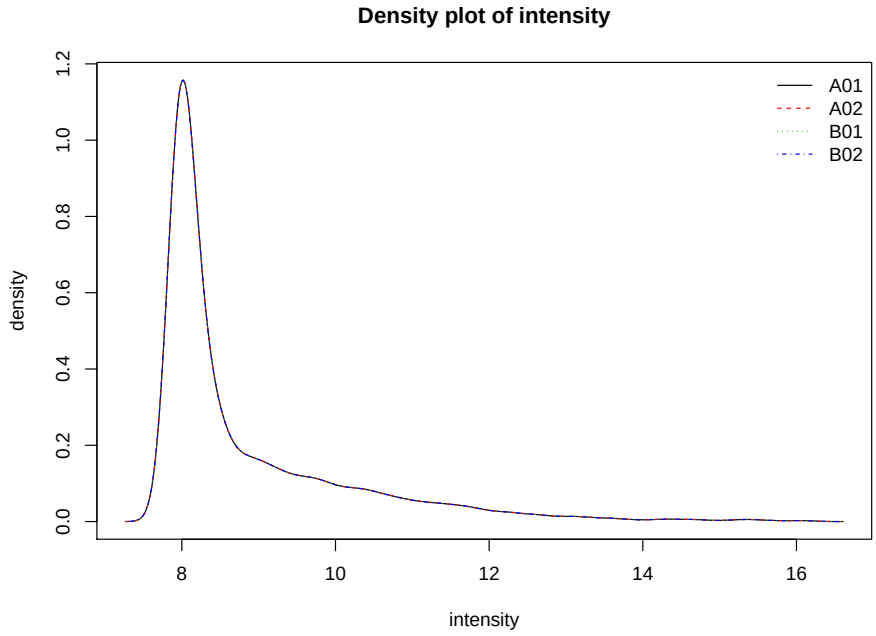


Figure 15: Density plot of Illumina microarrays after normalization

```
> plot(lumi.N.Q, what='boxplot')                    ## box plot
> # boxplot(lumi.N.Q)
```

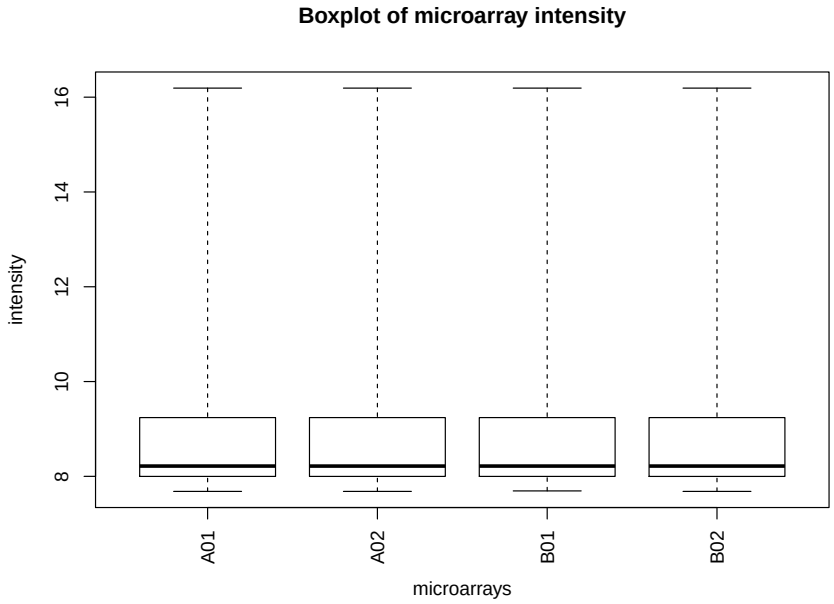


Figure 16: Density plot of Illumina microarrays after normalization

```
> plot(lumi.N.Q, what='pair')                    ## pairwise plots
```

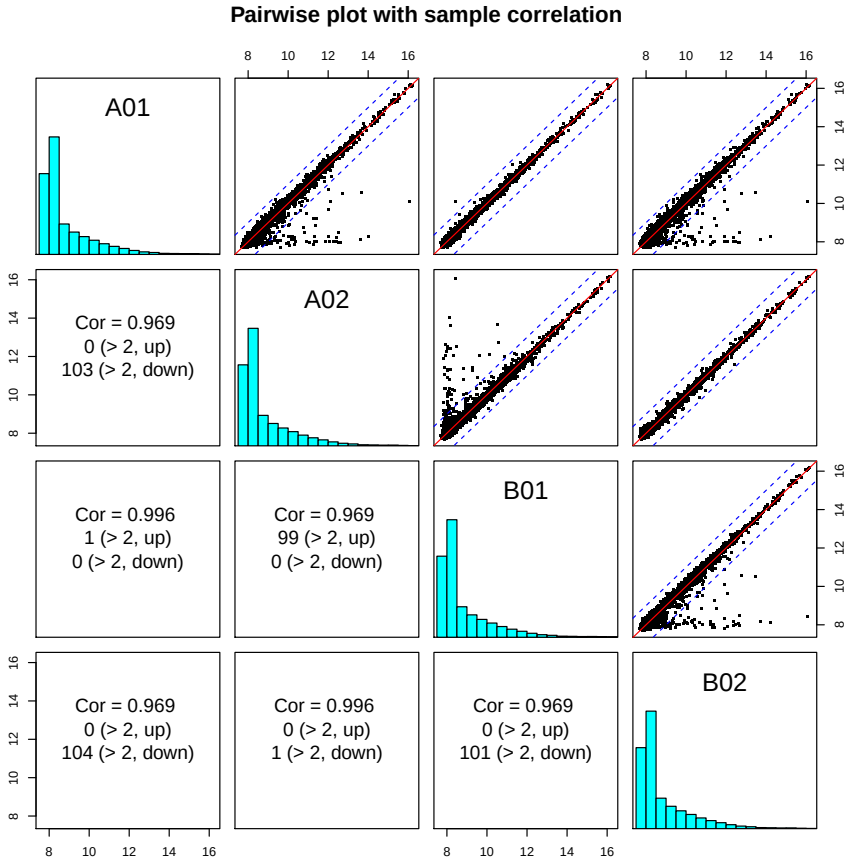


Figure 17: Pairwise plot with microarray correlation after normalization

```
> plot(lumi.N.Q, what='MAplot')                    ## plot the pairwise MAplot
```

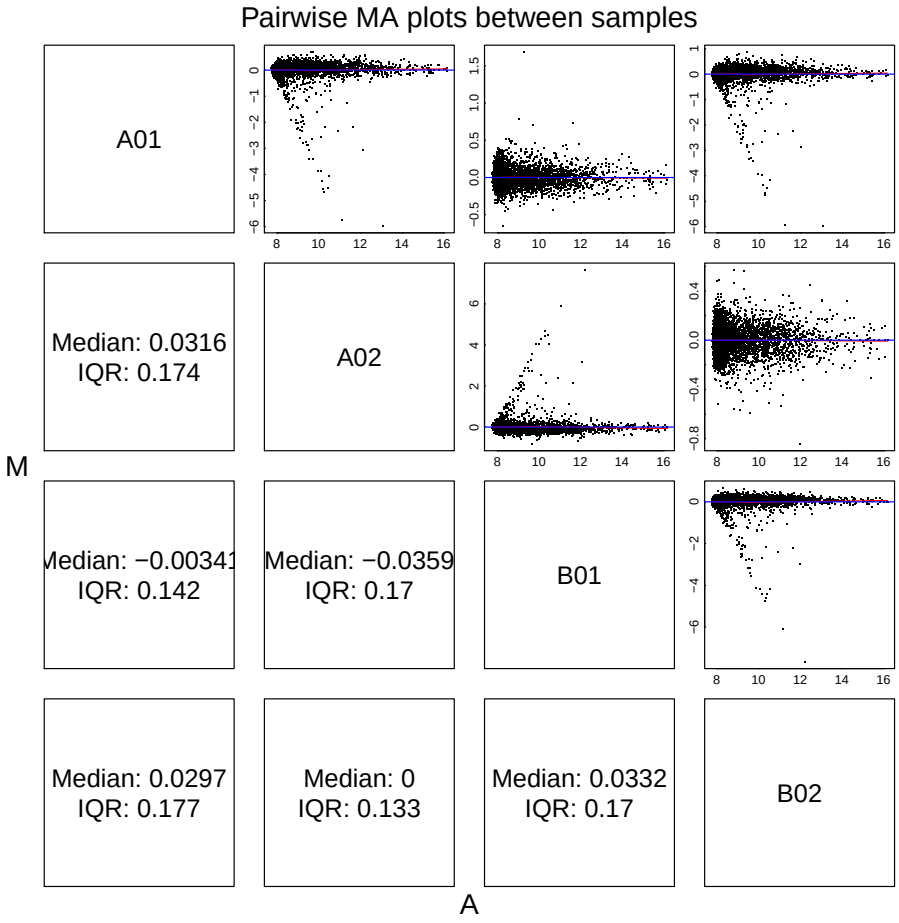


Figure 18: Pairwise MAplot after normalization

```
> ## plot the sampleRelation using hierarchical clustering
> plot(lumi.N.Q, what='sampleRelation')
```

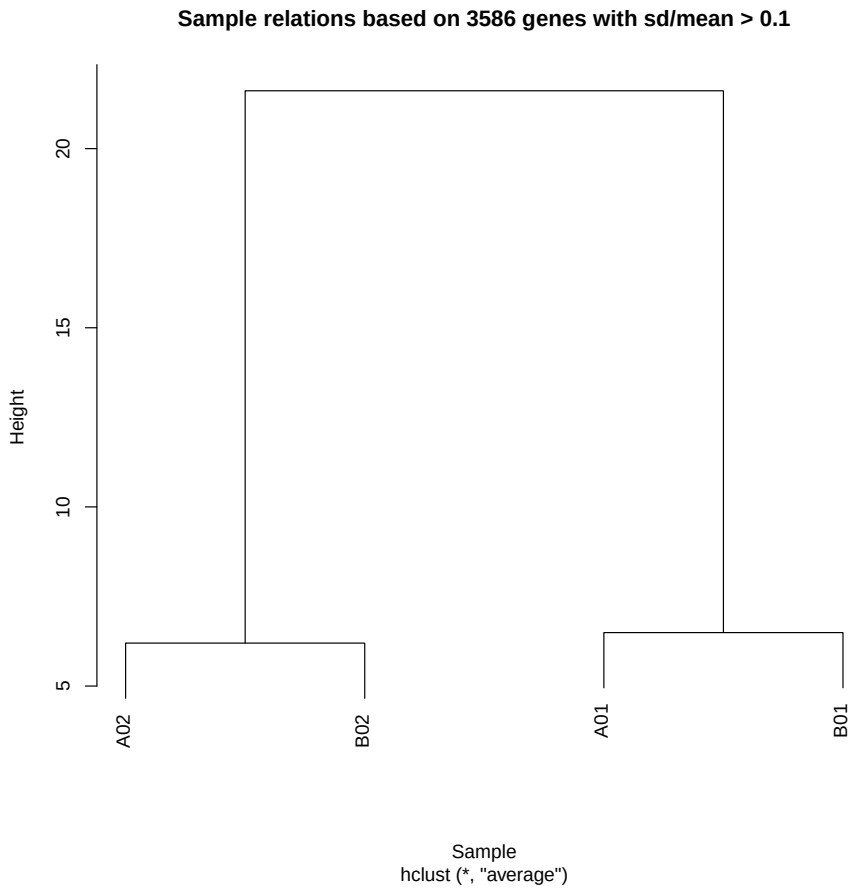


Figure 19: Sample relations after normalization

```
> ## plot the sampleRelation using MDS
> plot(lumi.N.Q, what='sampleRelation', method='mds', color=c("01", "02", "01", "02"))
```

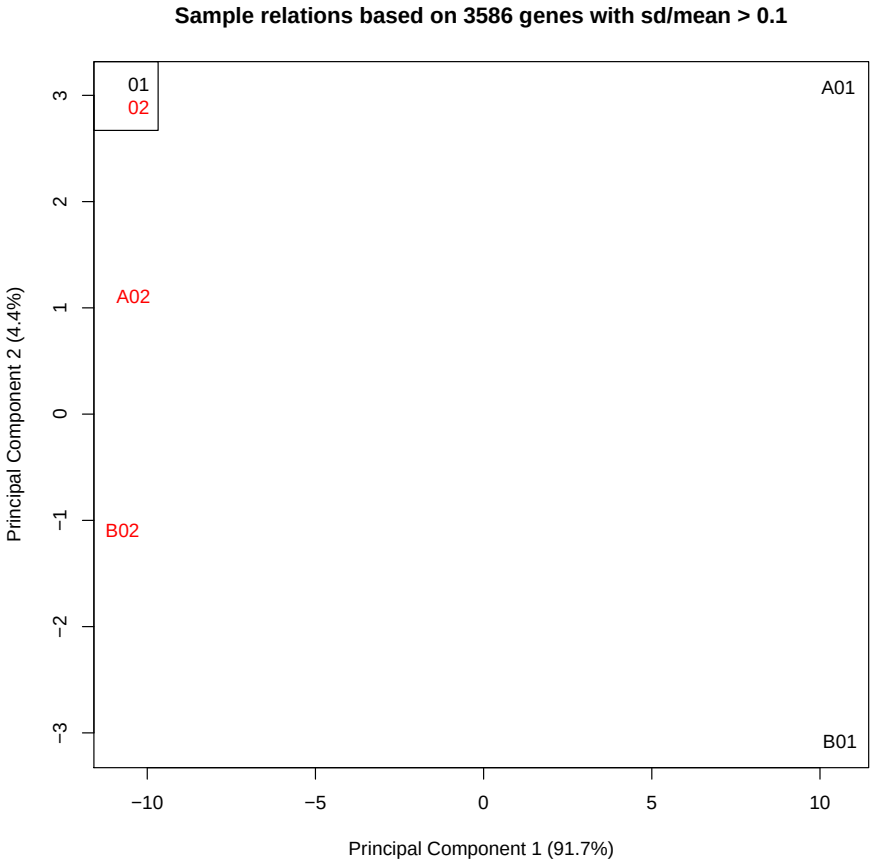


Figure 20: Sample relations after normalization

```
Background Correction: bgAdjust
Variance Stabilizing Transform method: vst
Normalization method: quantile


Background correction ...
Perform bgAdjust background correction ...
done.

Variance stabilizing ...
Perform vst transformation ...
2016-05-03 18:57:58 , processing array  1
2016-05-03 18:57:58 , processing array  2
2016-05-03 18:57:58 , processing array  3
2016-05-03 18:57:58 , processing array  4
done.

Normalizing ...
Perform quantile normalization ...
done.

Quality control after preprocessing ...
Perform Quality Control assessment of the LumiBatch object ...
done.
```

Users can easily customize the processing parameters. For example, if the user wants to do "rsn" normalization, the user can run the following code. For more details, please read the help document of `lumiExpresso` function.

```
> ## Do all the preprocessing with customized settings
> # lumi.N.Q <- lumiExpresso(example.lumi, normalize.param=list(method='rsn'))
```

5.8 Inverse VST transform to the raw scale

Figure **??** shows VST is very close to log2 in the high expression range. In convenience, users usually can directly use `2^x` to approximate the data in raw scale and estimate the fold-change. For the users concern more in the low expression range, we also provide the function `inverseVST` to resume the data in the raw scale. Need to mention, the inverse transform should be performed after statistical analysis, or else it makes no sense to transform back and forth. The `inverseVST` function can directly applied to the **LumiBatch** object after `lumiT` with VST transform, or VST transform plus RSN normalization (default method of `lumiN`). For the RSN normalized data, the inverse transform is based on the parameters of the Target Array because the Target Array is the benchmark data and is not changed after normalization. Other normalization methods, like quantile or loess, will change the values of all the arrays. As a result, no inverse VST transform available for them. Users may use some kind of approximation for the quantile normalized data by themselves. Here we just provide some examples of VST parameters retrieving and inverse VST transform.

```
> ## Parameters of VST transformed LumiBatch object
> names(attributes(lumi.T))

 [1] "history"           "controlData"       "QC"
 [4] "assayData"         "phenoData"         "featureData"
 [7] "experimentData"    "annotation"        "protocolData"
[10] ".__classVersion__" "class"             "vstParameter"
[13] "transformFun"

> ## VST parameters: "vstParameter"  and  "transformFun"
> attr(lumi.T, 'vstParameter')

           a           b        g Intercept
A01 0.4944205 0.010158205 1.461128  5.484722
A02 1.4778404 0.009076935 1.502390  5.349984
B01 2.4462693 0.009447733 1.541250  5.006017
B02 2.5012403 0.008940373 1.547270  5.048591

> attr(lumi.T, 'transformFun')

    A01     A02     B01     B02
"asinh" "asinh" "asinh" "asinh"

> ## Parameters of VST transformed and RSN normalized LumiBatch object
> names(attributes(lumi.N.Q))

 [1] "history"           "controlData"       "QC"
 [4] "assayData"         "phenoData"         "featureData"
 [7] "experimentData"    "annotation"        "protocolData"
[10] ".__classVersion__" "class"             "vstParameter"
[13] "transformFun"

> ## VSN "targetArray" , VST parameters: "vstParameter"  and  "transformFun"
> attr(lumi.N.Q, 'vstParameter')

          a           b           g   Intercept
2.419753559 0.009512344 1.514310826 5.197110136

> attr(lumi.N.Q, 'transformFun')

[1] "asinh"

> ## After doing statistical analysis of the data, users can recover to the raw scale for
> ## Inverse VST to the raw scale
> lumi.N.raw <- inverseVST(lumi.N.Q)
```

6 Handling large data sets

Several users asked about processing large data set, e.g., over 100 samples. Directly handling such big data set usually will cause "out of memory" error in most computers. In this case, when read the BeadStudio output file, we can ignore the "beadNum" (related columns. The function `lumiR` provides a parameter

called "columnNameGrepPattern". we can set the string grep pattern of "detection" and "beadNum" as NA. You can also ignore "detection" columns. However, the "detection" information is useful for the estimation of present count of each probe and used in the VST parameter estimation. To further save memory, you can suppress the input of annotation data by setting "inputAnnotation" as FALSE.

Here is some example code:

```
## load the data with empty detection and beadNum slots, and without annotation informatio
>  x.lumi <- lumiR("fileName.txt", columnNameGrepPattern=list(beadNum=NA), inputAnnotation
```

Usually, the large data set is composed of many small data files. In this case, the transformations, like log2 and `vst`, can be performed right after the input of each data file and some information can be removed in the object after transformation. *lumi* provides the `lumiR.batch` function for this purpose.

Here is some example code:

```
## load the list of data files (a vector of file names)
##        and do VST transformation for each file and combine the results.
>  x.lumi <- lumiR.batch(fileList, transform='vst')
```

Another good news is that the normalization, like `rsn` and `ssn` in the *lumi* package, can sequentially process the data and handle such large data set.

The solution can be like this:

1. Read the data file by smaller batches (e.g. 10 or just one by one), and then do the variance stabilization for each data batch using `lumiR.batch` or `lumiR` function.

2. Pick one sample as the target array for normalization and then using "RSN" or "SSN" normalization method to normalize all batches of data using the same target array.

3. Combine the normalized data. (In order to save memory, the user can first remove those probes not expressed in all samples.)

In the `rsn` and `ssn` functions, there is a parameter called "targetArray", which is the model for other chips to normalize. It can be a column index, a vector or a LumiBatch object with one sample. In our case, we need to use one LumiBatch object with one sample as the "targetArray". The selection of the target array is flexible. We suggest to choose the one most similar to the mean of all samples. For convenience, we can also just select the first sample as "targetArray" (suppose it has no quality problem). The selected target array will also be used for all other data batches. Since different data batches use the same target array as model, the results are comparable and can be combined!

Here is the example code:

```
## Read in the Batch ith data file, suppose named as "fileName.i.txt"
> x.lumi.i <- lumiR("fileName.i.txt")
## variance stabilization (using vst or log2 transform)
> x.lumiT.i <- lumiT(x.lumi.i)
## select the "targetArray"
## This target array will also be used for other batches of data.
## For convenience, here we just select the first sample as targetArray.
> targetArray <- x.lumiT.i[,1]
## Do RSN normalization
> x.lumiN.i <- lumiN(x.lumiT.i, method='rsn', targetArray=targetArray)
```

The normalized data batches can be combined by using function `combine(x, y)`.

7 Performance comparison

We have selected the Barnes data set [4], which is a series dilution of two tissues at five different dilutions, to compare different preprocessing methods. In order to better compare the algorithms, we selected the samples with the smallest dilution difference (the most challenging comparison), i.e., the samples with the dilution ratios of 100:0 and 95:5 (each condition has two technical replicates) for comparison. For the Barnes data set, because we do not know which of the signals are coming from 'true' differentially expressed genes, we cannot use an ROC curve to compare the performance of different algorithms. Instead, we evaluated the methods based on the concordance of normalized intensity profile and real dilution profile of the selected probes. More detailed evaluations with other criteria and based on other data sets can by found in our paper [2].

Following Barnes et al. (2005)[4], we defined a concordant gene (really a concordant probe) as a signal from a probe with a correlation coefficient larger than 0.8 between the normalized intensity profile and the real dilution profile (five dilution ratios with two replicates at each dilution ratio). If a selected differentially expressed probe is also a concordant one, it is more likely to be truly differentially expressed. Figure **??** shows the percentage of concordant probes among the selected probes, which were selected by ranking the probes' p-value (calculated based on *limma* package) from low to high. We can see the VST transformed data outperforms the Log2-transformed and VSN processed data. For the normalization methods, RSN and quantile normalization have similar performance for the VST transformed data, and RSN outperforms quantile for the Log transformed data.

Please see another vignette in the lumi package: `"lumi_vST_evaluation.pdf"` for more details of the evaluation of VST (Variance Stabilizing Transformation).

8 Gene annotation

One challenge of Illumina microarray is the inconsistency and changes of Illumina identifiers across versions, even across different releases. This makes the integration of the Illumina data difficult. In order to resolve these problems, we invented a nuID (nucleotide universal IDentifier) annotation system, released related annotation packages and a website to provide identifier mapping and the latest annotation. Please refer to the separate document ("Resolve the Inconsistency of Illumina Identifiers through nuID Annotation") in the lumi package for more details.

9 A use case: from raw data to functional analysis

Figure **??** shows the data processing flow chart of the use case. Since the classes in *lumi* package are inherited from class **ExpressionSet**, packages and functions

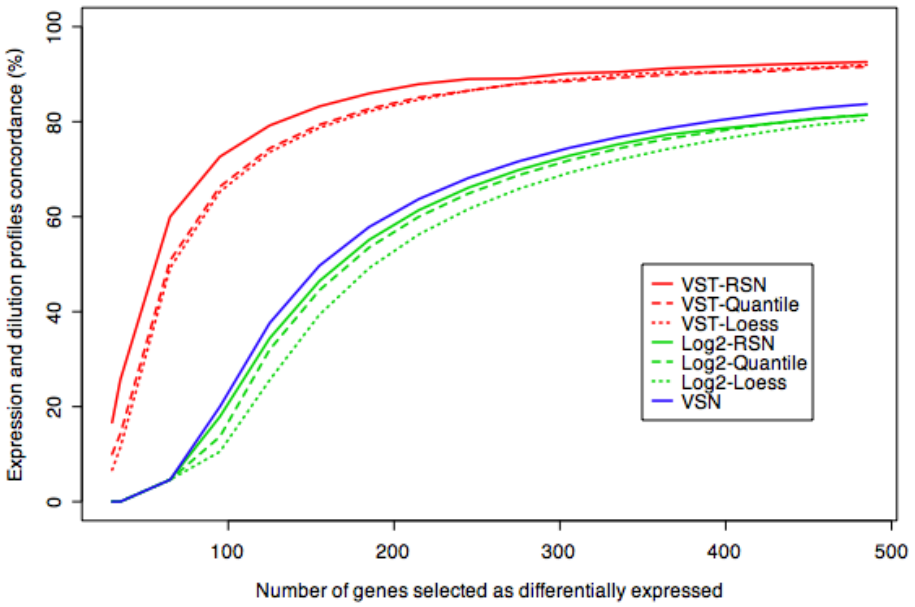


Figure 21: Comparison of the concordance between the expression and dilution profiles of the selected differentially expressed genes

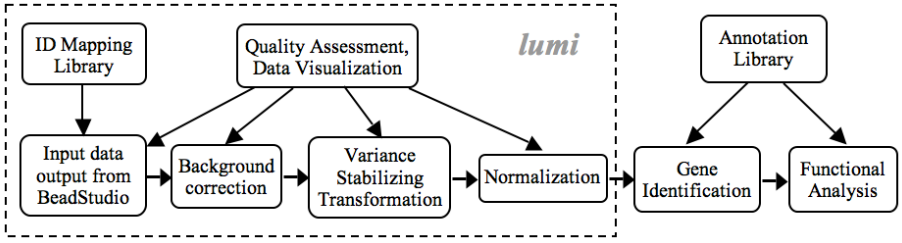


Figure 22: Flow chart of the use case

compatible with class **ExpressionSet** or accepting matrix as input all can be used for *lumi* results. Here we just give two examples: using *limma* to identify differentiated genes and using *GOstats* to annotate the significant genes.

We use the Barnes data set [4] as an example, which has be created as a Bioconductor experiment data package *lumiBarnes*. The Barnes data set measured a dilution series of two human tis-sues, blood and placenta. It includes six samples with the titration ratio of blood and placenta as 100:0, 95:5, 75:25, 50:50, 25:75 and 0:100. The samples were hybridized on HumanRef-8 BeadChip (Illumina, Inc) in duplicate. We select samples with titration ratio, 100:0 and 95:5 (each has two technique replicates) in this data set to evaluate the detection of differential expressions.

9.1 Preprocess the Illumina data

```
> library(lumi)
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'  #  Not run
> ## load the data if you want to provide the sample Information and map Illumina IDs to n
> # example.lumi <- lumiR.batch(fileName, lib.mapping='lumiHumanIDMapping', sampleInfoFile

> ## load saved data
> data(example.lumi)
> ## sumary of the daa
> example.lumi
> ## summary of quality control information
> summary(example.lumi, 'QC')

> ## preprocessing and quality control after normalization
> lumi.N.Q <- lumiExpresso(example.lumi, QC.evaluation=TRUE)
> ## summary of quality control information after preprocessing
> summary(lumi.N.Q, 'QC')

> ## Output the data as Tab separated text file
> write.exprs(lumi.N.Q, file='processedExampleData.txt')
```

9.2 Identify differentially expressed genes

Identify the differentiated genes based on moderated t-test using *limma*.

Retrieve the normalized data

```
> dataMatrix <- exprs(lumi.N.Q)
```

To speed up the processing and reduce false positives, remove the unexpressed and un-annotated genes

```
> presentCount <- detectionCall(example.lumi)
> selDataMatrix <- dataMatrix[presentCount > 0,]
> if (require(lumiHumanAll.db) & require(annotate)) {
+   selDataMatrix <- selDataMatrix[!is.na(getSYMBOL(rownames(selDataMatrix), 'lumiHumanAll
+ }
> probeList <- rownames(selDataMatrix)

> ## Specify the sample type
> sampleType <- c('100:0', '95:5', '100:0', '95:5')
> if (require(limma)) {
+         ##  compare '95:5' and '100:0'
+         design <- model.matrix(~ factor(sampleType))
+         colnames(design) <- c('100:0', '95:5-100:0')
+         fit <- lmFit(selDataMatrix, design)
+         fit <- eBayes(fit)
+         ## Add gene symbols to gene properties
+         if (require(lumiHumanAll.db) & require(annotate)) {
+                geneSymbol <- getSYMBOL(probeList, 'lumiHumanAll.db')
```

```
+                geneName <- sapply(lookUp(probeList, 'lumiHumanAll.db', 'GENENAME'), func
+                fit$genes <- data.frame(ID= probeList, geneSymbol=geneSymbol, geneName=ge
+           }
+         ## print the top 10 genes
+         print(topTable(fit, coef='95:5-100:0', adjust='fdr', number=10))
+
+         ## get significant gene list with FDR adjusted p.values less than 0.01
+         p.adj <- p.adjust(fit$p.value[,2])
+         sigGene.adj <- probeList[ p.adj < 0.01]
+         ## without FDR adjustment
+         sigGene <- probeList[ fit$p.value[,2] < 0.01]
+ }
```

```
                                    ID geneSymbol
ol_iQkR.siio.kvH6k ol_iQkR.siio.kvH6k      PLAC4
WlCoF7taz2MeYf3l6I WlCoF7taz2MeYf3l6I       SDC1
6QNThLQLd61eU6IXhI 6QNThLQLd61eU6IXhI       PSG9
EY761AIG0XSLUfnuyc EY761AIG0XSLUfnuyc        CGA
NSjRKdq2eSGf0ur4aQ NSjRKdq2eSGf0ur4aQ       PRG2
QaYYojcJJvVElV3I98 QaYYojcJJvVElV3I98       DLK1
uioiKiIlzFXx8k5EC4 uioiKiIlzFXx8k5EC4        CRH
TueuSaiCheWBxB6B18 TueuSaiCheWBxB6B18      KISS1
rSU1F9I7txuZ3lPQdo rSU1F9I7txuZ3lPQdo        DCN
Q.oCSr13l5wQlRuhS0 Q.oCSr13l5wQlRuhS0      FSTL1

ol_iQkR.siio.kvH6k
WlCoF7taz2MeYf3l6I
6QNThLQLd61eU6IXhI                                                              pregnancy sp
EY761AIG0XSLUfnuyc                                                              glycoprotein
NSjRKdq2eSGf0ur4aQ proteoglycan 2, bone marrow (natural killer cell activator, eosinophil
QaYYojcJJvVElV3I98                                                                    delta
uioiKiIlzFXx8k5EC4                                                                      cor
TueuSaiCheWBxB6B18
rSU1F9I7txuZ3lPQdo
Q.oCSr13l5wQlRuhS0
                       logFC   AveExpr        t      P.Value    adj.P.Val
ol_iQkR.siio.kvH6k 5.963373 10.660061 67.06413 1.268937e-13 3.563735e-10
WlCoF7taz2MeYf3l6I 5.242754 10.008626 63.23725 2.168749e-13 3.563735e-10
6QNThLQLd61eU6IXhI 4.934807  9.971604 60.12509 3.436320e-13 3.563735e-10
EY761AIG0XSLUfnuyc 6.359559 10.923226 59.34802 3.869097e-13 3.563735e-10
NSjRKdq2eSGf0ur4aQ 4.904642 10.198395 59.21052 3.951802e-13 3.563735e-10
QaYYojcJJvVElV3I98 4.622227 10.067377 57.22067 5.397012e-13 3.787205e-10
uioiKiIlzFXx8k5EC4 4.840778  9.878324 56.68580 5.879449e-13 3.787205e-10
TueuSaiCheWBxB6B18 5.082683 10.012350 55.26484 7.410384e-13 4.176678e-10
rSU1F9I7txuZ3lPQdo 4.592143  9.734708 52.64127 1.154380e-12 5.783444e-10
Q.oCSr13l5wQlRuhS0 4.433009 10.357602 51.59204 1.386823e-12 6.253186e-10
                         B
ol_iQkR.siio.kvH6k 21.72848
WlCoF7taz2MeYf3l6I 21.27740
6QNThLQLd61eU6IXhI 20.88015
```

```
EY761AIG0XSLUfnuyc 20.77635
NSjRKdq2eSGf0ur4aQ 20.75779
QaYYojcJJvVElV3I98 20.48214
uioiKiIlzFXx8k5EC4 20.40575
TueuSaiCheWBxB6B18 20.19792
rSU1F9I7txuZ3lPQdo 19.79439
Q.oCSr13l5wQlRuhS0 19.62537
```

Based on the significant genes identified using *limma* or t-test, we can do further analysis, like GO analysis (*GOstats* package) and machine learning (*MLInterface* package). Next, we will use GO analysis as an example.

9.3 Gene Ontology and other functional analysis

Based on the interested genes identified using *limma* or other tests, we can further do functional analysis. We can use package *GOstats*, *Category* and other packages to do this type of analysis.

There is one important thing need to mention during this type of analysis. As we described in previous section, the lumi annotation packages are nuID indexed and are built by pooling all types of chips of the same species. This makes it different from the traditional Affymetrix annotation packages, which is one package for one type of chip. Because lots of methods in Category/GOstats were originally designed based on the Affymetrix annotation packages, the default setting of these function may not work well for lumi annotation packages. However, this can be solved by first transferring the probe ids as Entrez Ids, and then do analysis at the Entrez Id level instead of the probe Id level. Please see our example for how to transfer Probe Ids as Entrez Ids.

Following is an example of performing Hypergeometric test of Gene Ontology based on the significant gene list (for e. Table **??** shows the significant GO terms of Molecular Function with p-value less than 0.01. Here only show the significant GO terms of BP (Biological Process). For other GO categories MF(Molecular Function) and CC (Cellular Component), it just follows the same procedure.

```
> if (require(GOstats) & require(lumiHumanAll.db)) {
+
+         ## Convert the probe Ids as  Entrez Ids and make them unique
+         sigLL <- unique(unlist(lookUp(sigGene,'lumiHumanAll.db','ENTREZID')))
+         sigLL <- as.character(sigLL[!is.na(sigLL)])
+         params <- new("GOHyperGParams",
+               geneIds= sigLL,
+               annotation="lumiHumanAll.db",
+               ontology="BP",
+               pvalueCutoff= 0.01,
+               conditional=FALSE,
+               testDirection="over")
+
+          hgOver <- hyperGTest(params)
+
+         ## Get the p-values of the test
+         gGhyp.pv <- pvalues(hgOver)
```

```
+
+         ## Adjust p-values for multiple test (FDR)
+         gGhyp.fdr <- p.adjust(gGhyp.pv, 'fdr')
+
+         ## select the Go terms with adjusted p-value less than 0.01
+         sigGO.ID <- names(gGhyp.fdr[gGhyp.fdr < 0.01])
+
+         ## Here only show the significant GO terms of BP (Molecular Function)
+         ##         For other categories, just follow the same procedure.
+         sigGO.Term <- getGOTerm(sigGO.ID)[["BP"]]
+ }
```

	GO ID	Term	p-value	Significant Genes No.	Total Genes No.
1	GO:0009611	response to wound...	8.4244e-06	42	443
2	GO:0006955	immune response	8.8296e-06	68	859
3	GO:0006952	defense response	1.7525e-05	72	945
4	GO:0006950	response to stres...	1.9132e-05	81	1103
5	GO:0009607	response to bioti...	5.0811e-05	72	976
6	GO:0009613	response to pest,...	7.2813e-05	45	533
7	GO:0006954	inflammatory resp...	0.00025402	25	250
8	GO:0009605	response to exter...	0.00026005	46	580
9	GO:0051707	response to other...	0.00040553	45	575
10	GO:0051674	localization of c...	0.00082563	30	348
11	GO:0006928	cell motility	0.00082563	30	348
12	GO:0040011	locomotion	0.00099205	30	352

Table 1: GO terms, p-values and counts.

9.4 GEO submission of the data

As more and more journals require the microarray data should be submitted to GEO website, we created GEO submission functions for users convenience. The submission file will be in the SOFT format. So users can submit all the data in a batch. There are two major functions users need to use. `produceGEOSampleInfoTemplate` produces a template of sample information with some default fillings. Some fields have been filed in with default settings. Users should fill in or modify the detailed sample descriptions by referring some previous submissions. No blank fields are allowed. Users are also suggested to fill in the "Sample_platform_id" by checking information of the GEO Illumina platform. `produceGEOSubmissionFile` is the main function of produce GEO submission file including both normalized and raw data information in the SOFT format. By default, the R objects, lumiNormalized, lumiRaw and sampleInfo, will be saved in a file named 'supplementaryData.Rdata'. (See help information of `produceGEOSubmissionFile`) Users can include this R data file as a GEO supplementary data file.

```
## Produce the sample information template
> produceGEOSampleInfoTemplate(lumi.example, lib.mapping = 'lumiHumanIDMapping', fileName
```

```
## After editing the 'GEOsampleInfo.txt' by filling in sample information
> produceGEOSubmissionFile(lumi.N.Q, lumi.example, lib='lumiHumanIDMapping', sampleInfo='G
```

10 Session Info

```
> toLatex(sessionInfo())
```

- R version 3.3.0 RC (2016-04-26 r70550), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.34.0, Biobase 2.32.0, BiocGenerics 0.18.0, IRanges 2.6.0, S4Vectors 0.10.0, XML 3.98-1.4, annotate 1.50.0, limma 3.28.0, lumi 2.24.0, lumiHumanAll.db 1.22.0, lumiHumanIDMapping 1.10.1, org.Hs.eg.db 3.3.0
- Loaded via a namespace (and not attached): BiocInstaller 1.22.0, BiocParallel 1.6.0, Biostrings 2.40.0, DBI 0.4, GEOquery 2.38.0, GenomeInfoDb 1.8.0, GenomicAlignments 1.8.0, GenomicFeatures 1.24.0, GenomicRanges 1.24.0, KernSmooth 2.23-15, MASS 7.3-45, Matrix 1.2-6, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.4.5, Rsamtools 1.24.0, SummarizedExperiment 1.2.0, XVector 0.12.0, affy 1.50.0, affyio 1.42.0, base64 1.1, beanplot 1.2, biomaRt 2.28.0, bitops 1.0-6, bumphunter 1.12.0, chron 2.3-47, codetools 0.2-14, colorspace 1.2-6, data.table 1.9.6, digest 0.6.9, doRNG 1.6, foreach 1.4.3, genefilter 1.54.0, grid 3.3.0, illuminaio 0.14.0, iterators 1.0.8, lattice 0.20-33, locfit 1.5-9.1, magrittr 1.5, matrixStats 0.50.2, mclust 5.2, methylumi 2.18.0, mgcv 1.8-12, minfi 1.18.0, multtest 2.28.0, nleqslv 3.0.1, nlme 3.1-127, nor1mix 1.2-1, pkgmaker 0.22, plyr 1.8.3, preprocessCore 1.34.0, quadprog 1.5-5, registry 0.3, reshape 0.8.5, rngtools 1.2.4, rtracklayer 1.32.0, siggenes 1.46.0, splines 3.3.0, stringi 1.0-1, stringr 1.0.0, survival 2.39-2, tools 3.3.0, xtable 1.8-2, zlibbioc 1.18.0

11 Acknowledgments

We would like to thanks the users and researchers around the world contribute to the lumi package, provide great comments and suggestions and report bugs. Especially, we would like to thanks Michal Blazejczyk, Peter Bram, Ligia Bras, Vincent Carey, Kevin Coombes, Sean Davis, Jean-Eudes DAZARD, Ryan Gordon, Wolfgang Huber, DeokHoon Kim, Matthias Kohl, Danilo Licastro, Ezhou Lori Long, Renee McElhaney, Martin Morgan, Ingrid H. G. stense, Denise Scholtens, Wei Shi, Gordon Smyth, Michael Stevens, Jiexin Zhang (sorted by last name) and many other people not mentioned here.

12 References

1. Du, P., Kibbe, W.A. and Lin, S.M., (2008) 'lumi: a pipeline for processing Illumina microarray', Bioinformatics 24(13):1547-1548

2. Lin, S.M., Du, P., Kibbe, W.A., (2008) 'Model-based Variance-stabilizing Transformation for Illumina Microarray Data', Nucleic Acids Res. 36, e11

3. Du, P., Kibbe, W.A. and Lin, S.M., (2007) 'nuID: A universal naming schema of oligonucleotides for Illumina, Affymetrix, and other microarrays', Biology Direct, 2, 16.

4. Barnes, M., Freudenberg, J., Thompson, S., Aronow, B. and Pav-lidis, P. (2005) "Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms", Nucleic Acids Res, 33, 5914-5923.