1 Introduction

This article introduces usage of the LMGene package. LMGene has been developed mainly for analysis of microarray data using a linear model and g-log data transformation in the R statistical package.

2 Data preparation

LMGene takes objects of class ExpressionSet, which is the standard data structure of the Biobase package. Hence, if data which is of class ExpressionSet already, the user can jump to further steps, like diagnostic plotting or g-log transformation. Otherwise, the user needs to generate new objects of class ExpressionSet. For more detail, please see the vignette, ‘Textual Description of Biobase’ in the Biobase package.

Note: ExpressionSet. In this package, an object of class ExpressionSet must produce proper data using the commands exprs(object) and phenoData(object).

Example. LMGene includes a sample array data which is of class ExpressionSet. Let’s take a look this sample data.

1. First, load the necessary packages in your R session.

   > library(LMGene)
   > library(Biobase)
   > library(tools)

2. Load the sample ExpressionSet class data in the package LMGene.
3. View the data structure of the sample data and the details of `exprs` and `phenoData` slots in the data.

```r
> slotNames(sample.eS)
[1] "assayData"  "phenoData"  "featureData"
[4] "experimentData" "annotation"  ".__classVersion__"
```

```r
> dim(exprs(sample.eS))
[1] 613 32
```

```r
> exprs(sample.eS)[1:3, ]
   p1d0 p1d1 p1d2 p1d3 p2d0 p2d1 p2d2 p2d3 p3d0 p3d1 p3d2 p3d3 p4d0 p4d1 p4d2
g1  216  149  169  113  193  172  167  185  162  246  227  173  151  179  142
g2  334  311  187  135  514  471  219  394  367  390  365  387  318  378  329
g3  398  367  351  239  712  523  356  629  474  438  532  427  429  574  419
   p4d3 p5d0 p5d1 p5d2 p5d3 p6d0 p6d1 p6d2 p6d3 p7d0 p7d1 p7d2 p7d3 p8d0 p8d1
g1  195  165  144  185  162  246  227  173  151  796  378  177  278  183  285
g2  450  293  285  428  645  631  324  343  852  451  259  379  259  386
g3  564  438  321  519  488  824  579  416  489 1046  501  375  388  373  509
   p8d2 p8d3
g1  275  202
g2  361  333
g3  468  436
```

```r
> phenoData(sample.eS)
```

An object of class "AnnotatedDataFrame"

```r
sampleNames: p1d0, p1d1, ..., p8d3 (32 total)  
```

varLabels and varMetadata description:

```r
patient: patient
```

```r
dose: dose
```

```r
> slotNames(phenoData(sample.eS))
[1] "varMetadata"  "data"  "dimLabels"
[4] ".__classVersion__"
```

Data generation. If you don’t have `ExpressionSet` class data, you need to make some. `LMGene` provides a function that can generate an object of class `ExpressionSet`, assuming that there are array data of `matrix` class and experimental data of `list` class.

1. The package has sample array and experimental data, `sample.mat` and `vlist`.  

2
> data(sample.mat)
> dim(sample.mat)

[1] 613 32

> data(vlist)
> vlist

$patient
[1] 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7 8 8 8 8 8
Levels: 1 2 3 4 5 6 7 8

$dose
[1] 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3

2. Generate ExpressionSet class data using neweS function.

> test.eS <- neweS(sample.mat, vlist)
> class(test.eS)

[1] "ExpressionSet"
attr("package")
[1] "Biobase"

> identical(sample.eS, test.eS)

[1] FALSE

c.f. If you have different types of array data, such as RGList, marrayRaw, and so on, you can convert them into ExpressionSet class by using as method after installing convert package.

3 G-log transformation

1. Estimating parameters for g-log transformation. The linear model is not applied to the raw data, but to transformed and normalized data. Many people use a log transform. LMGene uses a log-like transform involving two parameters. We estimate the parameters $\lambda$ and $\alpha$ of the generalized log transform $\log (y - \alpha + \sqrt{(y - \alpha)^2 + \lambda}) = \sinh^{-1} \left( \frac{y-\alpha}{\lambda} \right) + \log (\lambda)$ using the function tranest as follows:

> tranpar <- tranest(sample.eS)
> tranpar

$\lambda$
[1] 726.6187

$\alpha$
[1] 56.02754
The optional parameter ngenes controls how many genes are used in the estimation. The default is all of them (up to 100,000), but this option allows the use of less. A typical call using this parameter would be

```r
> tranpar <- tranest(sample.eS, 100)
> tranpar

$lambda
[1] 537.797

$alpha
[1] 47.50301
```

In this case, 100 genes are chosen at random and used to estimate the transformation parameter. The routine returns a list containing values for lambda and alpha.

2. G-log transformation. Using the obtained two parameters, the g-log transformed expression set can be calculated as follows.

```r
> trsample.eS <- transeS(sample.eS, tranpar$lambda, tranpar$alpha)
> exprs(sample.eS)[1:3, 1:8]

   p1d0  p1d1  p1d2  p1d3  p2d0  p2d1  p2d2  p2d3
g1  216   149   169   113   193   172   167   168
g2  334   311   187   135   514   471   219   394
g3  398   367   351   239   712   523   356   629

> exprs(trsample.eS)[1:3, 1:8]

   p1d0    p1d1    p1d2    p1d3    p2d0    p2d1    p2d2    p2d3
g1 5.824767 5.325979 5.502023 4.905114 5.679594 5.525992 5.485724 5.493906
g2 6.352509 6.269120 5.638029 5.181868 6.839016 6.742442 5.842253 6.542139
g3 6.553592 6.461210 6.409975 5.951666 7.192482 6.858102 6.426269 7.059150
```

3. Tranest options: multiple alpha, lowessnorm, model

Rather than using a single alpha for all samples, we can estimate a separate alpha for each sample. This allows for differences in chips, in sample concentration, or exposure conditions.

```r
> tranparmult <- tranest(sample.eS, mult = TRUE)
> tranparmult

$lambda
[1] 689.2819

$alpha
[1] 69.67146  37.02711  54.13904  69.35728  60.33270  60.75301  71.72965
[8] 64.55506  58.63427  65.73625  48.40173  59.43778  76.34568  78.81046
```
For vector alphas, transeS uses exactly the same syntax:

```r
> trsample.eS <- transeS(sample.eS, tranparmult$lambda, tranparmult$alpha)
> exprs(trsample.eS)[1:3, 1:8]
p1d0  p1d1  p1d2  p1d3  p2d0  p2d1  p2d2  p2d3
  g1  5.686954 5.424873 5.449682 4.549380 5.590642 5.418542 5.268332 5.347915
```

It's also possible to estimate the parameters using the more accurate lowess normalization (as opposed to uniform normalization):

```r
> tranparmult <- tranest(sample.eS, ngenes = 100, mult = TRUE, +  lowessnorm = TRUE)
> tranparmult

$lambda
 [1] 250.3344

$alpha
 [1] 80.48316 54.95829 61.94522 71.75155 70.93165 82.33413 87.45222
 [8] 78.43755 74.50446 72.55727 62.44456 68.08476 76.22532 88.81087
[15] 69.66951 95.44169 67.59351 61.22886 74.64438 76.00364 67.33350
[22] 77.70167 61.79198 63.21472 142.25690 119.34715 63.30083 87.58264
[29] 80.07982 95.93935 62.72441 62.99703
```

It is even possible now to estimate parameters using a specified model. For example, if we think that the interaction of variables in vlist is important, we can add interaction to the model:

```r
> tranpar <- tranest(sample.eS, model = "patient + dose + patient:dose")
> tranpar

$lambda
 [1] 860.0836

$alpha
 [1] 55.68625
```

The model is always specified in the same way as the right-hand side of an lm model. In the example above, we set the parameters to minimize the mean squared error for a regression of transformed gene expression against patient, log dose, and their interaction.
Be very careful of using interactions between factor variables. If you do not have enough replications, you can easily overfit the data and have no errors to work with.

Naturally, it’s possible to use mult, lowessnorm, and model all together.

4 Finding differentially expressed genes

1. Transformation and Normalization. Before finding differentially expressed genes, the array data needs to be transformed and normalized.

   \begin{verbatim}
   > trsample.eS <- transeS(sample.eS, tranparmult$lambda, tranparmult$alpha)
   > ntrsample.eS <- lnormeS(trsample.eS)
   \end{verbatim}

2. Finding differentially expressed genes The lmgene routine computes significant probes using the method of Rocke (2003). A typical call would be

   \begin{verbatim}
   > sigprobes <- LMGene(ntrsample.eS)
   \end{verbatim}

   There is an optional argument, level, which is the test level, .05 by default. A call using this optional parameter would look like

   \begin{verbatim}
   > sigprobes <- LMGene(ntrsample.eS, level = 0.01)
   \end{verbatim}

   The result is a list whose components have the names of the effects in the model. The values are the significant genes for the test of that effect or else the message "No significant genes".

   As with tranest, it’s possible to specify a more complex model to LMGene:

   \begin{verbatim}
   > sigprobes <- LMGene(ntrsample.eS, model = "patient+dose+patient:dose")
   > sigprobes
   \end{verbatim}

   $patient
   [1] "g2"  "g3"  "g9"  "g10" "g49"  "g54"  "g84"  "g85"  "g86"  "g93"
   [11] "g122" "g123" "g139" "g155" "g178" "g179" "g205" "g250" "g277" "g314"
   [21] "g319" "g327" "g336" "g353" "g372" "g375" "g384" "g399" "g405" "g406"
   [31] "g407" "g408" "g409" "g410" "g411" "g412" "g413" "g414" "g415" "g423"
   [41] "g425" "g426" "g461" "g462" "g463" "g477" "g503" "g520" "g524" "g566"
   [51] "g607"

   $dose
   [1] "No significant genes"

   $'patient:dose'
   [1] "No significant genes"

The routine LMGene requires the multtest package.
References


