

LMGene User's Guide

Geun-Cheol Lee, John Tillinghast, and David M. Rocke

April 16, 2015

Contents

1	Introduction	1
2	Data preparation	1
3	G-log transformation	3
4	Finding differentially expressed genes	5

1 Introduction

This article introduces usage of the **LMGene** package. **LMGene** has been developed for analysis of microarray data using a linear model and glog 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. Therefore, the user who already has data that is of class **ExpressionSet** can jump to further steps, such as g-log transformation or looking for differentially expressed genes. Otherwise, the user needs to generate new objects of class **ExpressionSet**. For more detail, please see the vignette, ‘An Introduction to Biobase and ExpressionSets’ 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 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**.

```
> data(sample.eS)
```

3. View the data structure of the sample data and the details of `exprs` and `phenoData` slots in the data.

```
> slotNames(sample.eS)

[1] "experimentData"      "assayData"           "phenoData"
[4] "featureData"         "annotation"           "protocolData"
[7] ".__classVersion__"
```

```
> dim(exprs(sample.eS))

[1] 613  32
```

```
> 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  168  151  179  142  156  160  214  157
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  390  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
```

```
> phenoData(sample.eS)

An object of class 'AnnotatedDataFrame'
 sampleNames: p1d0 p1d1 ... p8d3 (32 total)
 varLabels: patient dose
 varMetadata: labelDescription
```

```
> 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 includes sample array and experimental/phenotype data, `sample.mat` and `vlist`.

```
> 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
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"

```

3 G-log transformation

1. Estimating parameters for g-log transformation. In `LMGene`, the linear model is not intended to be 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 λ and α of the generalized log transform $\log(y - \alpha + \sqrt{(y - \alpha)^2 + \lambda}) = \sinh^{-1}(\frac{y - \alpha}{\sqrt{\lambda}}) + \log(\sqrt{\lambda})$ 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

```

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

```

```
$lambda
[1] 367.6927
```

```
$alpha
[1] 55.8105
```

In this case, 100 genes are chosen at random and used to estimate the transformation parameter. The function 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.

```
> 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	5.773068	5.238203	5.429309	4.766507	5.619359	5.455110	5.411737	5.420561
g2	6.322635	6.236562	5.575089	5.079337	6.820868	6.722415	5.791493	6.517556
g3	6.529296	6.434497	6.381818	5.906396	7.179810	6.840303	6.398580	7.044643

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.

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

```
$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
[15] 82.20326 96.19938 77.60070 79.48089 73.63257 73.41650 33.86029
[22] 69.26448 55.75460 54.29840 139.89493 91.36521 46.46158 59.02056
[29] 73.60255 89.48728 57.13887 64.98866
```

For vector alphas, transeS uses exactly the same syntax:

```
> trsample.eS <- transeS (sample.eS, tranparamult$lambda, tranparamult$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
g2	6.272797	6.308464	5.592073	4.915159	6.811348	6.710929	5.693269	6.492140
g3	6.488757	6.493737	6.388361	5.832776	7.173087	6.830052	6.345199	7.029530

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

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

```
$lambda
[1] 1000
```

```
$alpha
[1] 83 53 61 63 62 64 79 67 54 69 68 50 68 65 61 82 61 54 59
[20] 57 56 92 61 60 178 97 60 68 55 83 54 62
```

One may also specify a model other than the default no-interaction model. For example, if we think that the interaction of variables in `vlist` is important, we can add interaction to the model:

```
> 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 replicates, you can easily overfit the data and have no degrees of freedom left for error.

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.

```
> trsample.eS <- transeS (sample.eS, tranparamult$lambda, tranparamult$alpha)
> ntrsample.eS <- lnormeS (trsample.eS)
```

2. **Finding differentially expressed genes** The `LMGene` routine computes significant probes/genes by calculating gene-by-gene p-values for each factor in the model and adjusting for the specified false discovery rate (FDR). A typical call would be

```
> sigprobes <- LMGene(ntrsample.eS)
```

There is an optional argument, `level`, which is the FDR (default 5 percent). A call using this optional parameter would look like

```
> sigprobes <- LMGene(ntrsample.eS, level=.01)
```

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`:

```
> sigprobes <- LMGene(ntrsample.eS, model='patient+dose+patient:dose')
> sigprobes
```

```
$patient
 [1] "g2"   "g3"   "g4"   "g9"   "g10"  "g15"  "g43"  "g49"  "g54"  "g56"
[11] "g84"  "g85"  "g86"  "g88"  "g93"  "g123" "g155" "g176" "g178" "g179"
[21] "g183" "g240" "g277" "g304" "g305" "g310" "g336" "g375" "g399" "g405"
[31] "g406" "g407" "g408" "g409" "g411" "g412" "g413" "g414" "g415" "g417"
[41] "g423" "g461" "g462" "g463" "g477" "g485" "g503" "g516" "g520" "g528"
[51] "g544" "g566" "g607" "g612"

$dose
 [1] "No significant genes"

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

References

- [1] Benjamini, Y. and Hochberg, Y. (1995) "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society, Series B*, **57**, 289–300.
- [2] Durbin, B.P., Hardin, J.S., Hawkins, D.M., and Rocke, D.M. (2002) "A variance-stabilizing transformation for gene-expression microarray data," *Bioinformatics*, **18**, S105–S110.
- [3] Durbin, B. and Rocke, D. M. (2003a) "Estimation of transformation parameters for microarray data," *Bioinformatics*, **19**, 1360–1367.
- [4] Durbin, B. and Rocke, D. M. (2003b) "Variance-stabilizing transformations for two-color microarrays," *Bioinformatics*, **20**, 660–667.

- [5] Geller, S.C., Gregg, J.P., Hagerman, P.J., and Rocke, D.M. (2003) “Transformation and normalization of oligonucleotide microarray data,” *Bioinformatics*, **19**, 1817–1823.
- [6] Huber W., Von Heydebreck A., Sültmann H., Poustka A. and Vingron M. (2002) “Variance stabilization applied to microarray data calibration and to the quantification of differential expression,” *Bioinformatics*, **18**, S96–S104.
- [7] Rocke, David M. (2004) “Design and Analysis of Experiments with High Throughput Biological Assay Data,” *Seminars in Cell and Developmental Biology*, **15**, 708–713.
- [8] Rocke, D., and Durbin, B. (2001) “A model for measurement error for gene expression arrays,” *Journal of Computational Biology*, **8**, 557–569.
- [9] Rocke, D. and Durbin, B. (2003) “Approximate variance-stabilizing transformations for gene-expression microarray data,” *Bioinformatics*, **19**, 966–972.