LMGene User's Guide

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

December 14, 2007

Contents

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 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. Hence, if data which is `expressionSet` class is ready, the user can jump to further steps, like diagnostic plotting or g-log transformation. Otherwise, the user needs to generate new `expressionSet` class data. For more detail, please see the vignette, 'ExpressionSetIntroduction' in the `Biobase` package.
`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`.

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

3. View the data structure of the sample data and access data elements. You can obtain a brief summary of the contents of the ExpressionSet object by printing the object. You can extract data from it using a number of functions available. For example, you can extract the expression matrix and the phenotypic data using `exprs` and `phenoData`, respectively. Note that each column of the expression matrix has a different name.

```
> sample.eS

ExpressionSet (storageMode: lockedEnvironment)
assayData: 613 features, 32 samples
  element names: exprs
phenoData
  sampleNames: p1d0, p1d1, ..., p8d3  (32 total)
  varLabels and varMetadata description:
    patient: patient
    dose: dose
featureData
  featureNames: g1, g2, ..., g613  (613 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
Annotation: hgu95av2

> 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 and varMetadata description:
    patient: patient
    dose: dose
```

```
> pData(sample.eS)[1:4, ]

     patient dose
p1d0       1    0
p1d1       1    1
p1d2       1    2
p1d3       1    3
```

`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 `expressionSet` class, assuming that there are array data of `matrix` class and experimental data of `list` class. Note that the column names in the data matrix must be unique.

1. The package has sample array and experimental 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.

   ```
   > annotation = "hgu95av2"
   > test.eS <- neweS(sample.mat, vlist, annotation)
   > test.eS

   ExpressionSet (storageMode: lockedEnvironment)
   assayData: 613 features, 32 samples
     element names: exprs
   phenoData
     sampleNames: p1d0, p1d1, ..., p8d3  (32 total)
     varLabels and varMetadata description:
       patient: patient
       dose: dose
   featureData
     featureNames: g1, g2, ..., g613  (613 total)
     fvarLabels and fvarMetadata description: none
   experimentData: use 'experimentData(object)'
   Annotation: hgu95av2
   ```

`c.f.` If you have different types of array data, such as exprSet-like class, you may convert them into `ExpressionSet` class by using `as` method.

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 λ and α of the generalized log transform $\log\left(y - \alpha + \sqrt{(y-\alpha)^2 + \lambda}\right) = \sinh^{-1}(\frac{y-\alpha}{\lambda}) + \log(\lambda)$ 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

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

   $lambda
   [1] 749.5582

   $alpha
   [1] 55.68625
   ```

   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.

   ```
   > 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 5.777493 5.249972 5.437592 4.794392 5.625209 5.463010 5.420291 5.428978
g2 6.324307 6.238503 5.581431 5.095098 6.821592 6.723266 5.795773 6.518754
g3 6.530470 6.435876 6.383327 5.909877 7.180221 6.841005 6.400046 7.045150
```

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.

```
> 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
[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, 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
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):

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

$lambda
[1] 730.3523

$alpha
 [1]  86.49607  54.97201  61.78773  63.12011  66.36795  63.79551  76.69909
 [8]  65.41706  66.92179  64.34471  67.81681  61.79279  66.75633  69.73436
```

```
[15]  63.40896  87.59001  62.34851  56.59838  61.15531  63.59801  60.15077
[22]  92.50093  58.32517  58.68062 186.15555 107.16688  57.76507  72.15658
[29]  56.09527  84.31862  56.70260  65.55432
```

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:

```
> 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.

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

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

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

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

   ```
   > sigprobes <- LMGene(ntrsample.eS, level = 0.01)
   ```

   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"   "g9"   "g10"  "g14"  "g15"  "g49"  "g54"  "g84"  "g85"
[11] "g86"  "g93"  "g102" "g123" "g139" "g155" "g178" "g179" "g208" "g250"
[21] "g256" "g271" "g277" "g304" "g310" "g314" "g319" "g327" "g336" "g372"
[31] "g375" "g384" "g399" "g405" "g406" "g407" "g408" "g409" "g410" "g411"
[41] "g412" "g413" "g414" "g415" "g421" "g423" "g425" "g460" "g461" "g462"
[51] "g463" "g465" "g477" "g485" "g503" "g520" "g524" "g528" "g563" "g566"
[61] "g607" "g612"

$dose
[1] "No significant genes"

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

The routine LMGene requires the multtest package.

References

[1] 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.

[2] Durbin, B. and Rocke, D. M. (2003a) "Estimation of transformation parameters for microarray data," *Bioinformatics*, **19**, 1360–1367.

[3] Durbin, B. and Rocke, D. M. (2003b) "Exact and approximate variance-stabilizing transformations for two-color microarrays," submitted for publication.

[4] 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.

[5] Rocke, David M. (2004) "Design and Analysis of Experiments with High Throughput Biological Assay Data," *Seminars in Cell and Developmental Biology* , **15**, 708–713.

[6] Rocke, D., and Durbin, B. (2001) "A model for measurement error for gene expression arrays," *Journal of Computational Biology*, **8**, 557–569.

[7] Rocke, D. and Durbin, B. (2003) "Approximate variance-stabilizing transformations for gene-expression microarray data," *Bioinformatics*, **19**, 966–972.