Introduction to the les package: Identifying Loci of Enhanced Significance in Tiling Array Data

Julian Gehring

August 9, 2010

Abstract

In this vignette we describe using the *les* package for finding Loci of Enhanced Significance (LES) in tiling microarray data. With an example of a general framework we illustrate how to apply the package for exploring data to identify regions of differential regulation.

1 Introduction

Tiling microarrays have become an important platform for the investigation of regulation in expression and DNA-protein interaction. Due to their lower bias towards annotation compared to other microarray platforms they provide an powerful tool for biological research.

Beside the analysis of transcription for single conditions investigation of regulation between multiple experimental conditions is important for current research. A common approach consists in applying some suitable statistical tests on the level of single probes and thereby computing p-values p_i for each probe i independently. Since the targets of such experiments cover regions with several probes a reasonable next step involves combining information from neighboring probes. Many approaches use a smoothing window to obtain this. While these methods may work well most of them produce a feature that lacks a statistical interpretation.

In regions with differential effects the test statistics change their distribution and are referred to as *Loci of Enhanced Significance (LES)*. The changes of the test statistics depend on the underlying test applied.

The *les* package provides the ability to detect such *LES* independent of the underlying statistical test and can therefore be used for a wide range of applications. This vignette illustrates how *LES* can be identified with a general framework in tiling microarray data sets.

2 Data and statistics on probe level

For the analysis in this vignette we will use a simulated data set describing differential expression between two conditions. Please note that this data does not reflect real biological data. It is rather meant to illustrate the workflow than to provide real life data.

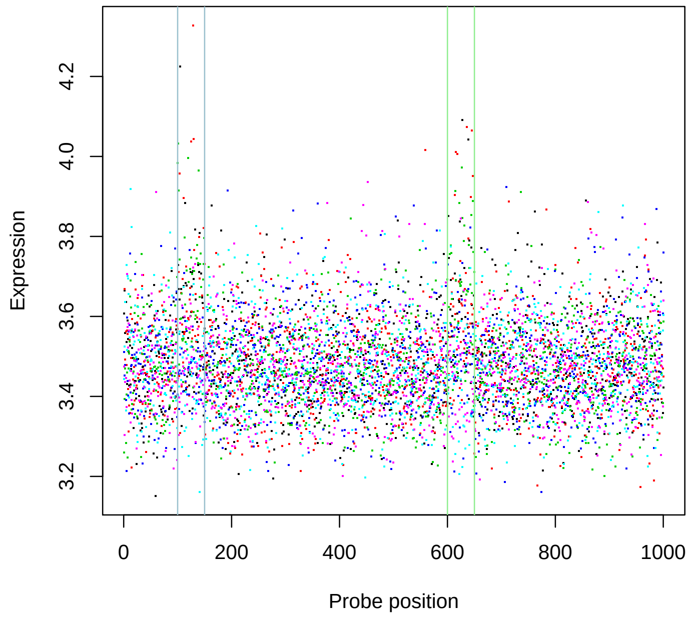
The data set consists of 1000 probes and two groups, e.g. treatment and control, each measured with three replicates. The expression values are stored in an expression set. We will extract the position of the probes, the conditions of the samples and the expression intensities. There are two regions of regulation present in the data each 50 bp long.

For experimental data, the positions of the probes are given by the design of the microarray. For simplicity we assume regularly spaced probes with 1 bp resolution in this illustration.

```
> library(les)
> library(Biobase)
> data(simTile)
> treatment <- as.factor(phenoData(simTile)$condition ==
+     "treatment")
> pos <- featureData(simTile)$position
> exprs <- exprs(simTile)
> regions <- c(100, 150, 600, 650)
> cols <- rep(c("lightblue3", "lightgreen"), each = 2)
```

Next we have a look at the expression values.

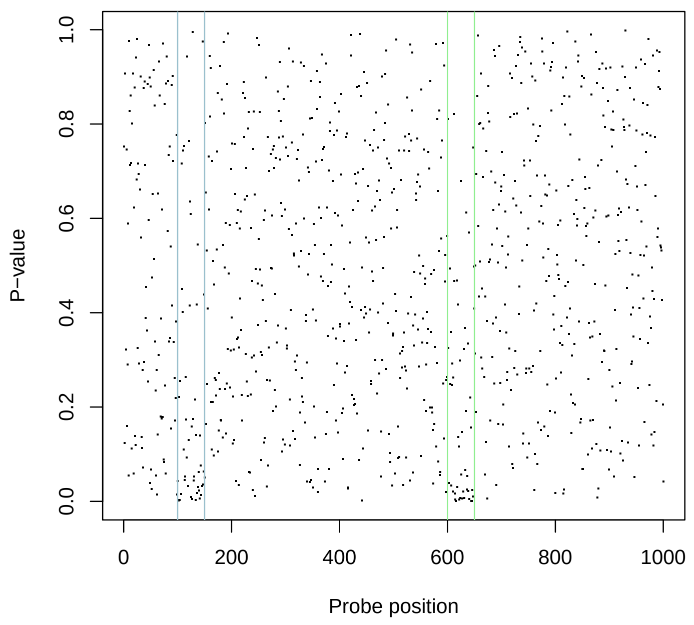
```
> matplot(exprs, pch = ".", xlab = "Probe position",
+     ylab = "Expression")
> abline(v = regions, col = cols)
```



In the next step we will compute compute a test statistics assessing regulatory changes between the two conditions for each probe. Since the sample size as for most tiling microarray experiments is small we will use a modified t-test provided by the *limma* package [2].

We will also plot the p-values p_i against the probe positions i.

```
> library(limma)
> design <- cbind(mean = 1, diff = treatment)
> fit <- lmFit(exprs, design)
> fit <- eBayes(fit)
> pval <- as.numeric(fit$p.value[, "diff"])

> plot(pos, pval, pch = ".", xlab = "Probe position",
+     ylab = "P-value")
> abline(v = regions, col = cols)
```



3 Incorporation information of neighboring probes

In tiling array experiments regions of regulation extend over several neighboring probes. Thereby p-values belonging to neighboring probes matching of the same region should contain mutual information and incorporation of such will be beneficial.

In the *les* package this in done in the following manner: For each probe i the surrounding p-values p_i get weights assigned by a windowing function. A weighted cumulative density is then computed and the fraction of significant p_i is estimated by iterative linear fitting. The method is based on the fact that p-values are uniformly distributed under the null hypothesis H_0 whereas p-values violating H_0 are shifted towards smaller values [1]. This results in Λ_i constituting an estimate of the fraction of p-values violating H_0 around the evaluated position and therefore the degree of significance in the local surrounding. It should be noted that this approach is closely related to the estimation of a false discovery rate and Λ_i can be interpreted in a similar way.

For the analysis we will first store our data in an object of class *Les* by calling the `create` function. The only data required for the analysis are the position of the probes i, the corresponding p-values p_i from the statistical test and optionally their chromosomal location.

```
> res <- create(pos, pval)
```

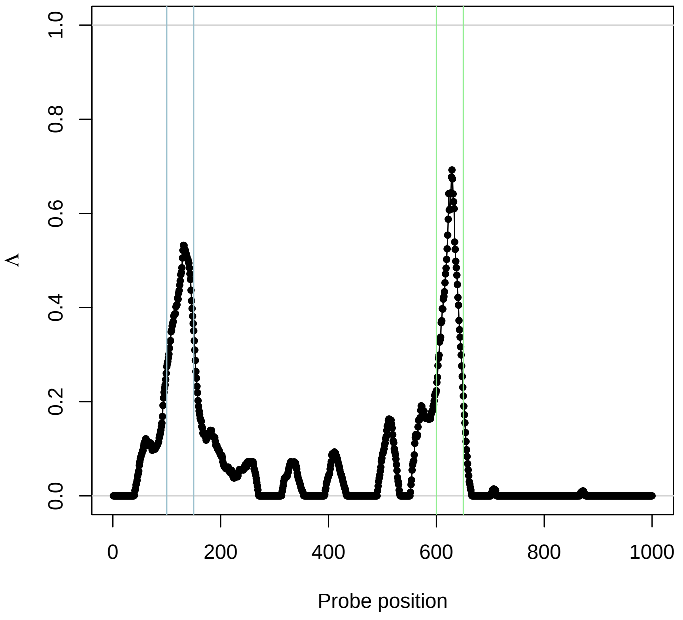
Then we can compute our first estimate of Λ_i for which we have to specify a window size. The power of detecting a region will be high if the size of the window matches approximately the size of the regulated region. In many experiments a rough prior knowledge on the region size is available which can be sufficient for choosing a window size.

By default a triangular weighting function will be chosen. We can also take different weighting functions such as a rectangular one or write our own function and pass it with the `weighting` argument. This step is described below in section 8 on page 11. Further we can specify whether we want to include the Grenander correction for the cumulative density or use multicore processing.

```
> win <- 30
> res <- estimate(res, win)
```

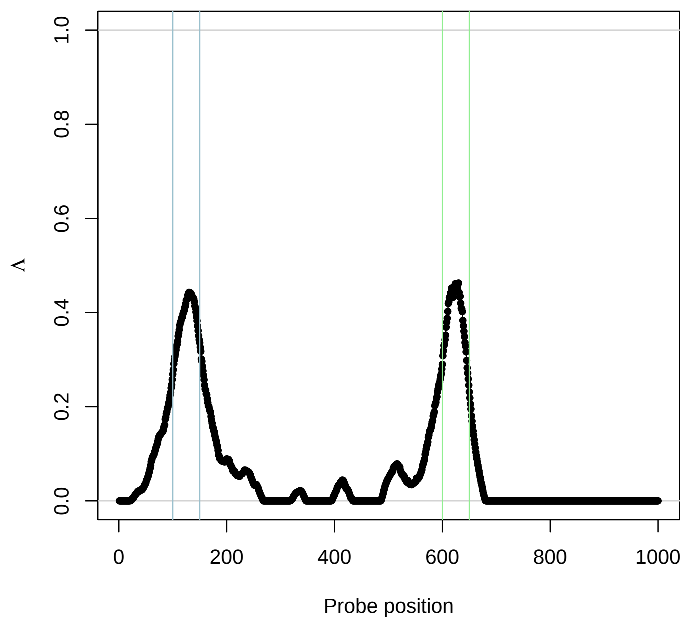
All data, results and parameters are now stored in the object named `res`. We can get a short summary on the results by calling *print*, *summary* or by plotting it.

```
> res

** Object of class 'Les' **
* 1000 probes on 1 chromosomes
* Lambda in range [0, 0.692297] with window size 30

> plot(res)
> abline(v = regions, col = cols)
```



For comparison we will analyze and plot the same data with a different window size. This allows us to explore our data set.

```
> win2 <- 50
> res <- estimate(res, win2)

> res

** Object of class 'Les' **
* 1000 probes on 1 chromosomes
* Lambda in range [0, 0.462787] with window size 50

> plot(res)
> abline(v = regions, col = cols)
```



We can already see two distinct peaks that correspond well to the simulated regions of regulation.

The *plot* method provides additional arguments that help customizing the figure.

4 Parameter estimation from the data

To turn the continuous Λ_i into distinct regions of interest we have to define a threshold Θ. It can be derived from the data by estimating the number of probes with a significant effect R on the whole array. Then Θ can be chosen such that $\mid \Lambda_i \geq \Theta \mid = R$. The content of any slot can be accessed by using the *[*-method.

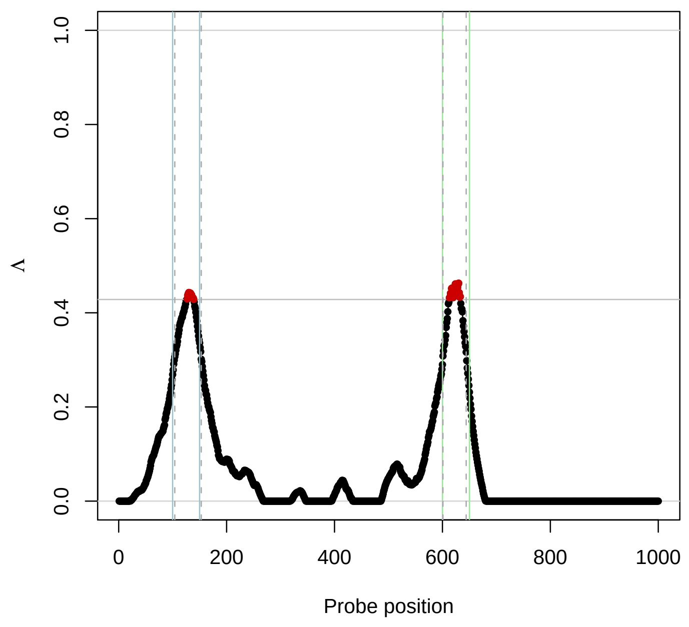
```
> res <- threshold(res, grenander = TRUE, verbose = TRUE)
```

```
[1] "34 significant probes estimated with limit Lambda>=0.42824"
```

Based on Θ we can look for regions that have a continuous $\Lambda_i \geq \Theta$. The *regions* method takes by default the estimated $\hat{\Theta}$ as shown before. We can also pass our own estimate for Θ with the `limit` argument. Further restrictions can be imposed on the regions such as the minimal length of a region and the maximum gap allowed between probes of one region.

A data frame with the estmated regions can be accessed with the *[*-method. This can also be used to access any other data slot of a `LES` object.

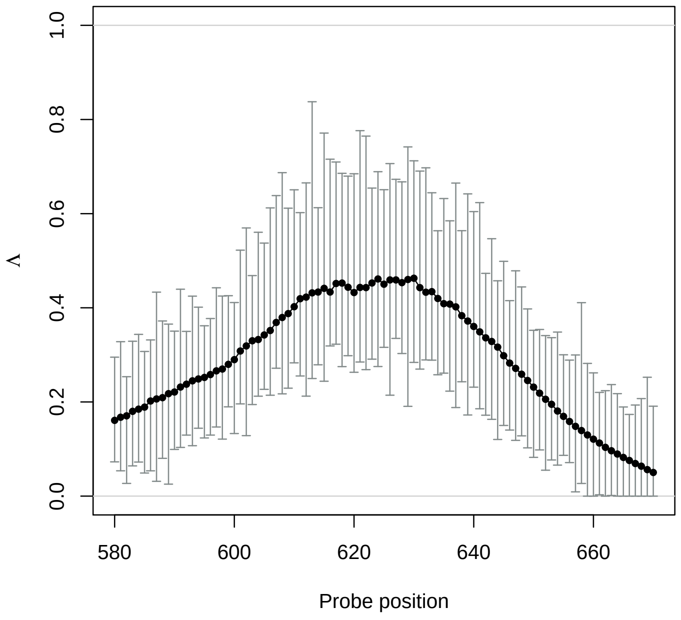
```
> theta <- 0.3
> res <- regions(res, limit = theta, verbose = TRUE)

[1] "2 regions found for lambda>=0.3"

> res

** Object of class 'Les' **
* 1000 probes on 1 chromosomes
* Lambda in range [0, 0.462787] with window size 50
* 35 regulated probes estimated for lambda >= 0.42824
* 2 regions detected

> res["regions"]

  chr start end size nProbes     ri     se      rs
1   0   104 153   50      50 0.4568 0.0342 13.3686
2   0   601 644   44      44 0.5616 0.0478 11.7400

> region <- res["regions"]
> borders <- c(region$start, region$end)
> plot(res)
> abline(v = regions, col = cols)
> abline(v = borders, col = "darkgray", lty = 2)
```



5 Calculation of confidence intervals

In some cases it is also useful to provide confidence intervals (CI) for Λ_i. These are computed by bootstrapping the probes in the window. Since bootstrapping is by its nature computationally demanding and CI are primarily interesting in regions of interest it it possible to compute CI for a subset of probes and to specify the number of bootstraps.

```
> subset <- pos >= 580 & pos <= 670
> res <- ci(res, subset, nBoot = 50)

> plot(res, error = "ci", limit = theta, xlim = c(580,
+     670))
```

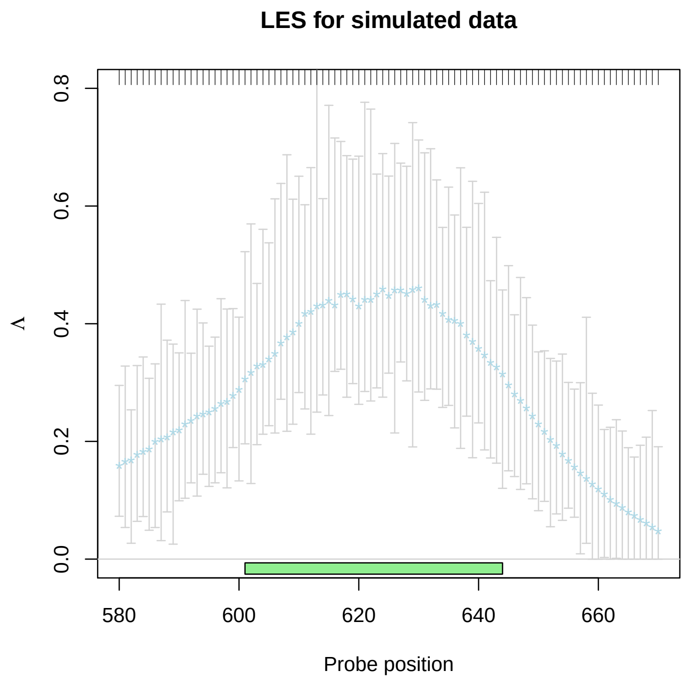


6 Plotting capabilities

The *plot* method provides many arguments for customizing figures.

The following command plots a smaller region of the chromosome with confidence intervals and the estimated region. Further the positions of the probes are shown at the top and the representation of the probes are changed.

```
> plot(res, error = "ci", region = TRUE, rug = TRUE,
+     rugSide = 3, main = "LES for simulated data",
+     probePch = "*", probeCol = "lightblue", errorCol = "lightgray",
+     regionCol = "lightgreen", xlim = c(580, 670),
+     ylim = c(0, 0.8), limit = FALSE, lty = 0)
```



7 Exporting result to external software

The estimated regions as well as Λ can be saved to a file with the *export* method. The regions can be exported to the *bed* and *gff* formats, the test statistic Λ to the *wig* format. These formats can be directly loaded into many genome software packages and browsers.

8 Specification of own window functions

With the `triangWeight`, `rectangWeight`, `epWeight` and `gaussWeight` four window functions are already included in the *les* package, providing a triangular, rectangular, Epanechnikov and Gaussian window, respectively. We can also specify own window functions and pass it via the `weighting` argument in the *estimate* method. They have to be specified in the following format, here illustrated with a triangular weighting.

```
> weightFoo <- function(distance, win) {
+     weight <- 1 - distance/win
+     return(weight)
+ }
> res2 <- estimate(res, 20, weighting = weightFoo)
```

References

[1] Kilian Bartholomé, Clemens Kreutz, and Jens Timmer. Estimation of gene induction enables a Relevance-Based ranking of gene sets. *Journal of Computational Biology*, 16(7):959–967, 2009.

[2] G. Smyth. limma: Linear models for microarray data. In *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, pages 397–420. 2005.

Session information

- R version 2.11.1 (2010-05-31), `x86_64-pc-linux-gnu`
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: Biobase 2.9.0, fdrtool 1.2.6, les 0.9.23, limma 3.5.15
- Loaded via a namespace (and not attached): annotate 1.27.1, AnnotationDbi 1.11.4, boot 1.2-42, DBI 0.2-5, gdata 2.8.0, genefilter 1.31.2, gplots 2.8.0, GSRI 1.1.7, gtools 2.6.2, RColorBrewer 1.0-2, RSQLite 0.9-2, splines 2.11.1, survival 2.35-8, xtable 1.5-6