# *blima* - R package for Bead Level Illumina Microarray Analysis

***Vojtěch Kulvait***[1]

[1] kulvait@gmail.com

**April 26, 2022**

## 1 Introduction

*blima* has been created for the bead (detector) level analysis of Illumina Microarray data. It is R/Bioconductor package. The package *blima* contains several functions implementing algorithms to preprocess Illumina Microarray data on the bead level. It also provides functions for probe level analysis and basic methods for differential expression testing.

For the background correction it contains implementation of background outlier correction method. From the standard methods there is implemented graphical background subtraction and RMA-like convolution model described in the work [1] as non parametric model. It implements variance stabilizing method and log transformation on the bead level to remove heteroskedasticity from the data. Then it also implements quantile normalization method for vectors of unequal lengths.

The *blima* uses the object *beadLevelData* from the package *beadarray* (see [2]) to store and manipulate with the data. *blima* functions take the *beadLevelData* object or list of such objects as input. By providing the list of such objects we can simply do the preprocessing based on multiple array kits. The *beadLevelData* contains records for each array and those arrays contains itself so called slots. These slots are storage units for bead level data on the array which we utilize.

To use this manual and follow examples, please also install the package *blimaTestingData* and load `blimatesting` object.

```
library(blima)
library(blimaTestingData)
data(blimatesting)
library(Biobase)

## Loading required package:  BiocGenerics

##
## Attaching package:  'BiocGenerics'

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, basename, cbind, colnames, dirname, do.call,
##     duplicated, eval, evalq, get, grep, grepl, intersect,
```

```
##     is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unsplit, which.max,
##     which.min

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'.  To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

library(xtable)
```

First we could print an overview of the data we have.

```
array1stats = chipArrayStatistics(blimatesting[[1]], includeBeadStatistic=TRUE,
        excludedOnSDMultiple=3)
array1pheno = pData(blimatesting[[1]]@experimentData$phenoData)
array1stats = data.frame(array1pheno$Name, array1stats)
colnames(array1stats)[1] <- "Array";
table = xtable(array1stats, align="|c|c|c|c|c|c|c|c|c|c|", caption="Array 1 statistic.")
digits(table)[c(2,3)]<-0
digits(table)[c(4:9)]<-1
print(table, include.rownames=FALSE)
```

| Array | Beads | Mean.FG | SD.FG | Mean.BG | SD.BG | Mean.BPP | SD.BPP | PCT |
|---|---|---|---|---|---|---|---|---|
| A1 | 1074567 | 808.2 | 513.7 | 698.2 | 1.7 | 22.2 | 5.4 | 0.50 |
| A2 | 1081885 | 804.0 | 523.7 | 698.4 | 1.7 | 22.4 | 5.5 | 0.54 |
| A3 | 1081138 | 811.5 | 536.9 | 697.9 | 1.7 | 22.4 | 5.3 | 0.58 |
| A4 | 1090648 | 803.2 | 485.0 | 698.0 | 1.7 | 22.6 | 5.5 | 0.20 |

**Table 1:** Array 1 statistic.

```
array2stats = chipArrayStatistics(blimatesting[[2]], includeBeadStatistic=TRUE,
        excludedOnSDMultiple=3)
array2pheno = pData(blimatesting[[2]]@experimentData$phenoData)
array2stats = data.frame(array2pheno$Name, array2stats)
colnames(array2stats)[1] <- "Array";
table = xtable(array2stats, align="|c|c|c|c|c|c|c|c|c|c|", caption="Array 2 statistic.")
digits(table)[c(2,3)]<-0
digits(table)[c(4:9)]<-1
print(table, include.rownames=FALSE)
```

| Array | Beads | Mean.FG | SD.FG | Mean.BG | SD.BG | Mean.BPP | SD.BPP | PCT |
|---|---|---|---|---|---|---|---|---|
| D4 | 1080439 | 818.5 | 573.0 | 698.1 | 2.7 | 22.4 | 5.4 | 0.04 |
| E1 | 1057564 | 809.9 | 527.8 | 697.7 | 1.7 | 21.9 | 5.2 | 0.53 |
| E2 | 1091118 | 822.0 | 577.7 | 697.9 | 1.7 | 22.6 | 5.5 | 0.30 |
| E3 | 1063812 | 811.0 | 523.0 | 697.8 | 1.8 | 22.0 | 5.2 | 0.28 |
| E4 | 1099047 | 806.6 | 546.0 | 697.8 | 1.7 | 22.7 | 5.3 | 0.49 |

**Table 2:** Array 2 statistic.

In the next sections we are going to provide you with a basic functionality of the blima package by performing simple task of exporting data to the format acceptable for NCBI Gene Expression Omnibus submission. In fact supporting data for this package in the GSE56129 dataset were prepared that way. The preprocessing shown in this manual is negative probes exclusion and background outlier correction with no other background correction step followed by variance stabilizing transformation (bead level) and quantile normalization (bead level for the vectors of unequal lengths). However in the manual there is also described how to utilize other preprocessing functions such as background subtraction, RMA like convolution and log2 transformation.

# 2 Data annotation

In the next chapters, it is necesery to have annotation data. Individual beads in the objects derived from the class *beadLevelData* are identified by "Array Address Id" more usual identifier in the databases and various sources is "Illumina Probe Id". So we would like to have the mapping from "Array Address Id" to "Illumina Probe Id". We also usually need an object to map "Illumina Probe Id" to "Symbol".

If we want to use background correction we will also need object to store "Array Address Ids" of negative control probes.

## 2.1 Working with Bioconductor annotation objects

First possibility is to use an annotation package such as *illuminaHumanv4.db*. In this case we can obtain objects with a mappings by following way

```
library(illuminaHumanv4.db)
```

```
## Loading required package:  AnnotationDbi
```

```
## Loading required package:  stats4
```

```
## Loading required package:  IRanges
```

```
## Loading required package:  S4Vectors
```

```
##
## Attaching package:  'S4Vectors'
```

```
## The following objects are masked from 'package:base':
##
##     I, expand.grid, unname
```

```
##
## Attaching package:  'IRanges'
```

```
## The following object is masked from 'package:grDevices':
##
##     windows
```

```
## Loading required package:  org.Hs.eg.db
```

```
##
```

```
##
```

```
adrToIllumina = toTable(illuminaHumanv4ARRAYADDRESS)
adrToIllumina = adrToIllumina[, c("ArrayAddress", "IlluminaID")]
colnames(adrToIllumina) = c("Array_Address_Id", "Probe_Id")
illuminaToSymbol = toTable(illuminaHumanv4SYMBOLREANNOTATED)
adrToSymbol = merge(adrToIllumina, illuminaToSymbol, by.x="Probe_Id", by.y="IlluminaID")
adrToSymbol = adrToSymbol[,c("Array_Address_Id", "SymbolReannotated")]
colnames(adrToSymbol) = c("Array_Address_Id", "Symbol")
negIl = mappedLkeys(revmap(illuminaHumanv4REPORTERGROUPNAME)["negative"])
negAdr = mappedRkeys(illuminaHumanv4ARRAYADDRESS[negIl])
```

## 2.2 Working with manufacturer annotation objects

Illumina annotation in the last subsection provides reanotated data meaning that some data may not match manufacturer annotation. For some applications like submission to Gene Expression Omnibus it should be necesery to work with default manufacturer annotation files. For this purpose is explained in the vignette for *blimaTestingData* package how to create annotation object called `annotationHumanHT12V4` which is R representation of text annotation provided by Illumina in file HumanHT-12_V4_0_R2_15002873_B.txt available for download from http://support.illumina.com/array/array_kits/humanht-12_v4_expression_beadchip_kit/downloads.ilmn. Provided you have prepared this object you can construct mappings by following way:

```
if(exists("annotationHumanHT12V4"))
{
    adrToIllumina = annotationHumanHT12V4$Probes[, c("Array_Address_Id", "Probe_Id")]
    adrToSymbol = annotationHumanHT12V4$Probes[, c("Array_Address_Id", "Symbol")]
    negAdr = unique(annotationHumanHT12V4$Controls[
       annotationHumanHT12V4$Controls$Reporter_Group_Name=="negative",
       "Array_Address_Id"])
}
```

# 3 Background correction

In the *blima* package I developed a special approach to the background correction of Illumina microarray data. In this approach there is no background subtraction. Instead we search for the beads for which the background value is out of range of 3 standard deviations from the mean of background values on the chip. We filter out these beads by creating slot with the value 1 for the beads passing background correction and 0 for those beads that do not pass.

Next we create background correction slot called "bgf" in the `blimatesting` object by calling function `bacgroundCorrect`. We also want to exclude any beads with negative values from the downstream analysis. This is possible by the function `nonPositiveCorrect`. We can create a new slot called "bgfnonnegative" but typically we would like to bitwise add these two channels. This addition is possible by setting parameter channelAndVector of functions `bacgroundCorrect` and `nonPositiveCorrect` to the name of the slot to perform logical and with. Our background correction code reads

```
blimatestingall = bacgroundCorrect(blimatesting, channelBackground = "GrnB",
        channelBackgroundFilter="bgf")
blimatestingall = nonPositiveCorrect(blimatestingall, normalizationMod=NULL,
```

```
        channelCorrect="GrnF",  channelBackgroundFilter="bgf", channelAndVector="bgf")
```

The blima package also contain methods for standard background correction. We can use the function `backgroundChannelSubtract` for background subtraction and `xieBacgroundCorrect` for RMA-like background convolution. If we want to create slot "BGS" for the graphical background subtracted data (subtract "GrnF - GrnB") we may call

```
blimatestingall = backgroundChannelSubtract(blimatestingall, normalizationMod = NULL,
        channelSubtractFrom = "GrnF", channelSubtractWhat = "GrnB", channelResult = "BGS")
```

There is also function for RMA-like background convolution. Method `xieBacgroundCorrect` do in fact use negative control probes information and is based on non parametric model described in the work [1]. To use this model we first have to prepare array addresses of negative control probes in the array. Then we can perform the convolution. If we want to create slot "XIE" processed by the algorithm we call

```
blimatestingall = xieBacgroundCorrect(blimatestingall, normalizationMod = NULL,
        negativeArrayAddresses=negAdr, channelCorrect="GrnF", channelResult="XIE",
        channelInclude="bgf")
```

# 4 Variance Stabilizing Method

Next we do the variance stabilizing step (based on the model from [3] modified for bead level analysis) by calling the function `varianceBeadStabilise`. We use the channel "GrnF" as a quality to stabilize, we include only those beads to vst analysis having channelInclude "bgf" equals 1 for a given bead and we produce a new channel channelOutput="vst".

```
blimatestingall = varianceBeadStabilise(blimatestingall, quality="GrnF",
        channelInclude="bgf", channelOutput="vst")
```

If we do like to perform log2 transformation instead of variance stabilization for example of the slot "GrnF" we can call function `selectedChannelTransform` with transformation parameter set to `log2TransformPositive` we call the result "LOG"

```
blimatestingall = selectedChannelTransform(blimatestingall, normalizationMod=NULL,
        channelTransformFrom="GrnF", channelResult="LOG",
        transformation=log2TransformPositive)
```

# 5 Quantile normalization

In the next step we perform the quantile normalization of the bead level data. We use the bead level quantile normalization algorithm implemented in the function `quantileNormalize`.

```
blimatestingall = quantileNormalize(blimatestingall, normalizationMod=NULL,
        channelNormalize="vst", channelOutput="qua", channelInclude="bgf")
```

# 6 Data testing

The package *blima* provides a basic infrastructure for performing bead level and probe level testing of the data by means of functions `doTTests` and `doProbeTTests`.

Here we show how to proceed if we want to know 10 most differentially expressed probes (and associate genes) between groups A and E. We know that in the object `blimatesting` from the *blimaTestingData* package there is one extra array. We have no intention to include this array into the analysis. By means of the parameter `normalizationMod` of the functions `bacgroundCorrect`, `nonPositiveCorrect`, `varianceBeadStabilise` and `quantileNormalize` we can choose a subset of the arrays in our list of *beadLevelData* objects for the further processing. The `normalizationMod` specifies a list of logical vectors with the same structure as the first input object of given function, typically list of *beadLevelData* objects or single *beadLevelData* object. In the `normalizationMod` object the logical value TRUE at certain position means to process corresponding chip spot, logical value FALSE means to exclude corresponding chip slot from processing.

First we make logical lists corresponding to arrays to process and groups of arrays.

```
data("blimatesting")
groups1 = "A";
groups2 = "E";
sampleNames = list()
groups1Mod = list()
groups2Mod = list()
processingMod = list()
for(i in 1:length(blimatesting))
{
        p = pData(blimatesting[[i]]@experimentData$phenoData)
        groups1Mod[[i]] = p$Group %in% groups1;
        groups2Mod[[i]] = p$Group %in% groups2;
        processingMod[[i]] = p$Group %in% c(groups1, groups2);
        sampleNames[[i]] = p$Name
}
```

Then we process only those microarrays in the `processingMod` by our pipeline.

```
blimatesting = bacgroundCorrect(blimatesting, normalizationMod = processingMod,
                channelBackgroundFilter="bgf")
blimatesting = nonPositiveCorrect(blimatesting, normalizationMod = processingMod,
                channelCorrect="GrnF",  channelBackgroundFilter="bgf", channelAndVector="bgf")
blimatesting = varianceBeadStabilise(blimatesting, normalizationMod = processingMod,
                quality="GrnF", channelInclude="bgf", channelOutput="vst")
blimatesting = quantileNormalize(blimatesting, normalizationMod = processingMod,
                channelNormalize="vst", channelOutput="qua", channelInclude="bgf")
```

Then we do the tests. We have to decide what a good measure of differential expression is. In this example we choose $(1 - p_{adjusted})|FC|$. Then we list 10 most differentially expressed probes with corresponding genes according to this measure.

```
probeTest <- doProbeTTests(blimatesting, groups1Mod, groups2Mod,
                transformation=NULL, quality="qua", channelInclude="bgf")
beadTest <- doTTests(blimatesting, groups1Mod, groups2Mod,
```

```
                    transformation=NULL, quality="qua", channelInclude="bgf")
probeTestID = probeTest[,"ProbeID"]
beadTestID = beadTest[,"ProbeID"]
probeTestFC = abs(probeTest[,"mean1"]-probeTest[,"mean2"])
beadTestFC = abs(beadTest[,"mean1"]-beadTest[,"mean2"])
probeTestP = probeTest[,"adjustedp"]
beadTestP = beadTest[,"adjustedp"]
probeTestMeasure = (1-probeTestP)*probeTestFC
beadTestMeasure = (1-beadTestP)*beadTestFC
probeTest = cbind(probeTestID, probeTestMeasure)
beadTest = cbind(beadTestID, beadTestMeasure)
colnames(probeTest) <- c("ArrayAddressID", "difexPL")
colnames(beadTest) <- c("ArrayAddressID", "difexBL")
tocmp <- merge(probeTest, beadTest)
tocmp = merge(tocmp, adrToSymbol, by.x="ArrayAddressID", by.y="Array_Address_Id")
tocmp = tocmp[, c("ArrayAddressID", "Symbol", "difexPL", "difexBL")]
sortPL = sort(-tocmp[,"difexPL"], index.return=TRUE)$ix
sortBL = sort(-tocmp[,"difexBL"], index.return=TRUE)$ix
beadTop10 = tocmp[sortBL[1:10],]
probeTop10 = tocmp[sortPL[1:10],]
beadTop10 = xtable(beadTop10, align="|c|c|c|c|c|", caption="Top 10 probes on bead level.")
probeTop10 = xtable(probeTop10, align="|c|c|c|c|c|", caption="Top 10 probes on probe level.")
digits(beadTop10)[2] = 0
digits(probeTop10)[2] = 0
print(beadTop10, include.rownames=FALSE)
```

| ArrayAddressID | Symbol | difexPL | difexBL |
|---|---|---|---|
| 6590132 | IGFBP3 | 3.82 | 3.84 |
| 6840372 | IGFBP3 | 3.56 | 3.56 |
| 730414 | APOE | 2.73 | 3.41 |
| 150373 | FABP4 | 1.93 | 3.29 |
| 5810685 | THBS1 | 3.18 | 3.27 |
| 2360326 | TAGLN | 1.84 | 3.23 |
| 6960142 | COL1A1 | 2.34 | 2.90 |
| 4900520 | SCG2 | 1.13 | 2.81 |
| 110181 | KIAA1199 | 2.13 | 2.73 |
| 4260139 | AKR1B1 | 2.40 | 2.73 |

**Table 3:** Top 10 probes on bead level.

```
print(probeTop10, include.rownames=FALSE)
```

You can see that the bead level analysis and the probe level analysis in fact do produce comparable results.

| ArrayAddressID | Symbol | difexPL | difexBL |
|---|---|---|---|
| 6590132 | IGFBP3 | 3.82 | 3.84 |
| 6840372 | IGFBP3 | 3.56 | 3.56 |
| 5810685 | THBS1 | 3.18 | 3.27 |
| 730414 | APOE | 2.73 | 3.41 |
| 7210632 | AKR1C3 | 2.47 | 2.61 |
| 610519 | TPM1 | 2.43 | 2.51 |
| 5560246 | TPM1 | 2.42 | 2.48 |
| 4260139 | AKR1B1 | 2.40 | 2.73 |
| 6960142 | COL1A1 | 2.34 | 2.90 |
| 1400446 | PLIN2 | 2.25 | 2.26 |

**Table 4:** Top 10 probes on probe level.

# 7 Summarization

Even though the *blima* provides methods to work with unsummarized data it may be necesery sometimes to work also with summarized data. In the particular application mentionned in this manual we export the data to the format accepted by Gene Expression Omnibus. We need to have data summarized according to the Illumina Probe ID. This summarization the data from any slot is implemented in the function `createSummarizedMatrix`.

We allready prepared `adrToIllumina` mapping thus we create summarized matrix by calling the `createSummarizedMatrix`. Then we translate the ArrayAddressID to the ProbeID and create the output matrces in the form acceptable by the Gene Expression Omnibus. We prepare two matrices, first based on the data from the "qua" slot and second based on the "GrnF" slot. The "qua" matrix is called normalized data and "GrnF" matrix is called non normalized data.

The data for dataset dataset GSE56129 was prepared in the similar fashion using `annotation HumanHT12V4` object.

```
nonnormalized = createSummarizedMatrix(blimatesting, spotsToProcess=processingMod, quality="GrnF", channelInc
        annotationTag="Name")
nonnormalized = merge(nonnormalized, adrToIllumina, by.x="ProbeID", by.y="Array_Address_Id")
nonnormalized = nonnormalized[, c(10, 2:9)]
colnames(nonnormalized)[1] = "ID_REF"
for(i in 2:9)
{
    colnames(nonnormalized)[i] = sprintf("%s", colnames(nonnormalized)[i])
}
table = head(nonnormalized)
table = xtable(table, align="|c|c|c|c|c|c|c|c|c|c|", caption="Head of nonnormalized data.")
digits(table)[c(2:9)]<-1
print(table, include.rownames=FALSE)
```

```
normalized = createSummarizedMatrix(blimatesting, spotsToProcess=processingMod, quality="qua", channelInclude
        annotationTag="Name")
normalized = merge(normalized, adrToIllumina, by.x="ProbeID", by.y="Array_Address_Id")
normalized = normalized[, c(10, 2:9)]
colnames(normalized)[1] = "ID_REF"
for(i in 2:9)
```

| ID_REF | A1 | A2 | A3 | A4 | E1 | E2 | E3 | E4 |
|---|---|---|---|---|---|---|---|---|
| ILMN_1802380 | 859.6 | 867.6 | 858.6 | 860.1 | 889.9 | 919.4 | 937.1 | 924.94 |
| ILMN_1893287 | 715.7 | 715.8 | 716.7 | 703.9 | 714.7 | 717.0 | 711.3 | 717.44 |
| ILMN_3238331 | 713.9 | 710.5 | 714.3 | 714.4 | 707.4 | 715.2 | 715.8 | 710.99 |
| ILMN_1736104 | 707.6 | 720.8 | 714.1 | 715.4 | 709.4 | 700.2 | 712.6 | 715.53 |
| ILMN_1792389 | 711.0 | 720.6 | 722.8 | 710.6 | 717.5 | 718.2 | 717.9 | 717.87 |
| ILMN_1854015 | 727.9 | 728.8 | 729.4 | 721.0 | 728.4 | 727.1 | 726.1 | 728.90 |

**Table 5:** Head of nonnormalized data.

```
{
    colnames(normalized)[i] = sprintf("%s", colnames(normalized)[i])
}
table = head(normalized)
table = xtable(table, align="|c|c|c|c|c|c|c|c|c|", caption="Head of normalized data.")
digits(table)[c(2:10)]<-3
print(table, include.rownames=FALSE)
```

| ID_REF | A1 | A2 | A3 | A4 | E1 | E2 | E3 | E4 |
|---|---|---|---|---|---|---|---|---|
| ILMN_1802380 | 9.320 | 9.401 | 9.305 | 9.332 | 9.418 | 9.415 | 9.515 | 9.560 |
| ILMN_1893287 | 8.774 | 8.769 | 8.781 | 8.694 | 8.772 | 8.781 | 8.751 | 8.783 |
| ILMN_3238331 | 8.765 | 8.743 | 8.770 | 8.770 | 8.724 | 8.773 | 8.776 | 8.748 |
| ILMN_1736104 | 8.723 | 8.795 | 8.769 | 8.775 | 8.746 | 8.693 | 8.759 | 8.773 |
| ILMN_1792389 | 8.744 | 8.799 | 8.810 | 8.736 | 8.785 | 8.787 | 8.785 | 8.785 |
| ILMN_1854015 | 8.821 | 8.840 | 8.839 | 8.789 | 8.840 | 8.824 | 8.825 | 8.842 |

**Table 6:** Head of normalized data.

# Acknowledgement

Thanks to Pavla Jumrová for the language corrections.

# References

[1] Yang Xie, Xinlei Wang, and Michael Story. Statistical methods of background correction for illumina BeadArray data. *Bioinformatics*, 25(6):751–757, March 2009. PMID: 19193732. URL: http://bioinformatics.oxfordjournals.org/content/25/6/751, doi:10.1093/bioinformatics/btp040.

[2] Mark J. Dunning, Mike L. Smith, Matthew E. Ritchie, and Simon Tavaré. beadarray: R classes and methods for illumina bead-based data. *Bioinformatics*, 23(16):2183 –2184, 2007. URL: http://bioinformatics.oxfordjournals.org/content/23/16/2183.abstract, doi:10.1093/bioinformatics/btm311.

[3] Simon M Lin, Pan Du, Wolfgang Huber, and Warren A Kibbe. Model-based variance-stabilizing transformation for Illumina microarray data. *Nucleic Acids Research*, 36(2):e11–e11, February 2008. URL: http://nar.oxfordjournals.org/content/36/2/e11.abstract, doi:10.1093/nar/gkm1075.