

limma: Preprocessing, exploratory analysis and quality assessment

Natalie P. Thorne

July 12, 2007

Quick review of basics

Exercise 1: Change to the swirl directory in the `limmaPreprocessing` folder. After preprocessing the swirl data, you will save the workspace for a later lab. Use the `targets` file to read in the swirl SPOT data. Get the gene annotation and spot types information for this data set. Find the printer layout and set the control status for each gene.

```
> library(limma)
> targets = readTargets()
> RG = read.maimages(targets$FileName, source = "spot")
> RG$targets = targets
> RG$genes = readGAL()
> RG$printer = getLayout(RG$genes)
> types = readSpotTypes()
> RG$status = controlStatus(types, RG)
```

Spot weights and other information

Exercise 2: Re-read in the swirl data and get extra spot information about the area of the spots in each array from the image output files. Also get the foreground pixel intensity summaries based on the median for each spot for each array. Make a plot showing the relationship between the spot area and spot intensity. Describe the relationship between the area and intensity of spots on these arrays? Above what value might you consider the spot area unusually large? We will use this value later to construct spot quality weights.

☞ Look at one of the SPOT output files for the different columns of information they contain. The additional columns of information you will read in will be the `area`, `Rmedian` and `Gmedian` columns; these will be stored into the `other` component of the `RGList` object.

☞ The information that can be stored in the `other` component of an `RGList` is determined by the columns of information stored in each image output file. `other` stores spot information for each array (i.e. matrices with columns corresponding to arrays and rows to spots). In very large data sets, it is not recommended that you read in too many extra columns of

information into the `other` component; it will take too long and the resulting `RGList` object may become too large.

☞ The function `MA.RG` was used to calculate the `M` and `A` values. Background correction is performed by default in this function. We will look at the issue of background correction more closely later in this lab.

```
> RGo = read.maimages(targets$FileName, other.columns = c("area", "Rmedian", "Gmedian"),
+   source = "spot")
> names(RGo)
> names(RGo$other)
> dim(RGo)
> dim(RGo$other$area)
> MAo = MA.RG(RGo)
> plot(MAo$A[, 1], RGo$other$area[, 1])
```

📖 You may wish to read in extra columns of data about each spot for each array for the purposes of quality assessment. Subsequent investigation may reveal this information to be useful in the analysis or interpretation of your data or in particular for assessing the quality of each spot on the array (perhaps leading to creating spot quality weights). In `limma` you can create weights for each spot on each array, depending on the quality or informativeness of each spot. The weights are used to make sure that poorer quality spots don't contribute as much to statistical calculations as good quality, informative spots. Weights should be values between 0 and 1. The default weights for each spot are 1, meaning that all spots are considered equally in any statistical analyses. A simple weights vector for an array might consist of binary values (0's or 1's) for *bad* and *good* spots respectively. This is effectively the same as filtering out spots. Whereas a more sophisticated vector of weights for an array might have a range of values between 0 and 1, meaning that the degree of quality for different spots varies.

♣ *Note that spot weights are relative, i.e. if all spots have weight of 0.3 then, because their relative weights are equal, none are downweighted. To downweight whole arrays see `arrayWeights`.*

📖 `limma` has three default methods for calculating spot weights. One function `wtarea` is designed for SPOT data, `wtflags` is designed for GenePix data and `wtIgnore.Filter` is designed for QuantArray data.

Exercise 3: Read the section in the `limma` User's Guide on Spot Quality Weights.

Exercise 4: Re-read in the swirl data, this time use the `wtarea` to calculate weights for your spots by how similar they are to the ideal area of a spot.

```
> RGw1 = read.maimages(targets$FileName, wt.fun = wtarea(150), source = "spot")
> show(RGw1)
```

Exercise 5: (Optional) Write a simple function to calculate weights based on the `badspot` flag column. Look in one of the swirl array output files and look at the various columns in the output from the SPOT image analysis program. `badspot` is one of the column names in

the header of spot files. Now, calculate weights to filter out spots if the foreground mean and median differ by a certain threshold. Make sure you hit ENTER at the end of each line of code as shown.

```
> myfun2 = function(x) {
+   as.numeric(x$badspot < 1)
+ }
> RGw2 = read.maimages(targets$FileName, wt.fun = myfun2, source = "spot")
> show(RGw2)
> myfun3 = function(x, threshold = 500) {
+   okred = abs(x$Rmean - x$Rmedian) < threshold
+   okgreen = abs(x$Gmean - x$Gmedian) < threshold
+   as.numeric(okred & okgreen)
+ }
> RGw3 = read.maimages(targets$FileName, wt.fun = myfun3, source = "spot")
> show(RGw3)
```

Exercise 6: Get the weights object from RGw1 and store it in RG. Remove the following objects that aren't needed anymore; RGo, RGw1, RGw2, RGw3, and MAo. (Obviously, you will only be able to remove RGw2 and RGw3 if you carried out **Exercise 5**.)

☞ remove and rm perform the same function.

```
> RG$weights = RGw1$weights
> names(RG)
> show(RG)
> dim(RG)
> dim(RG$weights)
> objects()
> remove(RGo)
> rm(RGw1)
> rm(RGw2)
> rm(RGw3)
> rm(MAo)
> objects()
```

Diagnostic plots

Exercise 7: Make MA-plots of the four swirl arrays with and without the control genes highlighted. Try plotting with and without the legend. Use the for function to loop through each array number, consecutively making an MA-plot for each array. Also try using plotMA3by2; look up the help on this function to find out what it does.

```
> plotMA(RG)
> par(mfrow = c(2, 2))
> plotMA(RG, array = 1)
```

```

> plotMA(RG, array = 2)
> plotMA(RG, array = 3)
> plotMA(RG, array = 4)
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+   plotMA(RG, array = i)
+ }
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+   plotMA(RG, array = i, status = RG$Status)
+ }
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+   plotMA(RG, array = i, status = RG$Status, legend = FALSE)
+ }
> plotMA3by2(RG)

```

Exercise 8: Make spatial plots of the red and green background for the first array. Use the `low` and `high` arguments in `imageplot` to adjust the colour range of the spatial plots. Then make multiple figure plots for all arrays. Use the `ask` argument in the `par` function to control the nature of plotting over many pages. Also try using `imageplot3by2`; look up the help on this function to see what it does.

☞ When you use the argument `ask=TRUE` in the `par` function, it is necessary to click on the graphics window or to press ENTER in order to see the next set of plots.

```

> par(mfrow = c(1, 1))
> imageplot(RG$Rb[, 1], layout = RG$printer)
> imageplot(RG$Rb[, 1], layout = RG$printer, low = "white", high = "red")
> imageplot(RG$Rb[, 1], layout = RG$printer, low = "red", high = "darkred")
> par(mfrow = c(1, 2))
> imageplot(RG$Rb[, 1], layout = RG$printer, low = "white", high = "red")
> imageplot(RG$Gb[, 1], layout = RG$printer, low = "white", high = "green")
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+   imageplot(RG$Rb[, i], layout = RG$printer, low = "white", high = "red")
+   imageplot(RG$Gb[, i], layout = RG$printer, low = "white", high = "green")
+ }
> par(mfrow = c(2, 2), ask = TRUE)
> for (i in 1:4) {
+   imageplot(RG$Rb[, i], layout = RG$printer, low = "white", high = "red")
+   imageplot(RG$Gb[, i], layout = RG$printer, low = "white", high = "green")
+ }
> par(ask = FALSE)
> imageplot3by2(RG, "Gb")

```

☞ Note that the functions `plotMA3by2` and `imageplot3by2` might not work correctly if you are working with X Window System (X11) on Mac or Linux. If that is the case, make sure there is a proper installation of the PNG reference library (`libpng`, for instance) on your system.

☞ Also note that the `ask` argument in the `par` function might not work correctly if you are working with X Window System (X11) on Mac or Linux.

Exercise 9: Look at the distribution of foreground log-intensities in the red and green channels for each array using `plotDensities`. Consider whether the shapes of the densities are similar between red and green channels and between arrays.

```
> par(mfrow = c(1, 1))
> plotDensities(RG)
> plotDensities(RG, log = TRUE)
> plotDensities(RG, log = FALSE)
> plotDensities(RG, groups = c(1, 2, 3, 4), col = c("cyan", "orange", "magenta",
+         "blue"))
```

Exercise 10: Make boxplot summaries of the red and green foreground and background log-intensities for each slide. Notice whether the arrays with higher background also have higher foreground values. Are the red and green background levels similar? Which arrays have the highest background levels?

```
> par(mfrow = c(1, 1))
> boxplot(log2(RG$R) ~ col(RG$R), col = "red")
> fgbg = cbind(RG$R, RG$Rb, RG$G, RG$Gb)
> boxplot(log2(fgbg) ~ col(fgbg), col = c(rep("red", 8), rep("green", 8)))
```

Exercise 11: Look at the trend in foreground log-intensity values according to the order in which the spots were printed on the array. The function `plotPrintorder` may take a while to calculate the print order before plotting. Is there any trend in the overall intensity of spots according to the print order; is the trend similar between red and green channels and between slides?

☞ The `plotPrintorder` function may take a while to run.

```
> plotPrintorder(RG, layout = RG$printer)
> plotPrintorder(RG, layout = RG$printer, separate.channels = TRUE)
> plotPrintorder(RG, layout = RG$printer, separate.channels = TRUE, slide = 2)
```

Background correction

Exercise 12: Perform background correction on the raw data. Look up the help for `backgroundCorrect` and find out the options for the possible methods for background correction. Try performing no background correction, the default subtraction method and the minimum method.

☞ Background correction can be performed during the normalisation step. However, to investigate the different background correction methods separately from the normalisation methods, in this example we use the `backgroundCorrect` function in isolation.

```
> RG.nb = backgroundCorrect(RG, method = "none")
> RG.sb = backgroundCorrect(RG, method = "subtract")
> RG.mb = backgroundCorrect(RG, method = "minimum")
> names(RG)
> names(RG.nb)
> names(RG.sb)
> names(RG.mb)
```

Exercise 13: Use MA-plots to compare the data resulting from these different background correction methods for background measurements from SPOT. Save the MA-plots to a file in the current directory. Note that the command `pdf("file.pdf")` opens a file such that all subsequent plots to the graphics device will be written to this file rather than to the graphics window in R. Make sure you use `dev.off()` to close the file after making the plots you want to store to the file.

```
> pdf("MASwirl.pdf")
> par(mfrow = c(2, 2))
> plotMA(RG.nb, array = 2, xlim = c(0, 16), ylim = c(-5, 5), main = "bkgcorr: none")
> plotMA(RG.sb, array = 2, xlim = c(0, 16), ylim = c(-5, 5), main = "bkgcorr: subtract")
> plotMA(RG.mb, array = 2, xlim = c(0, 16), ylim = c(-5, 5), main = "bkgcorr: minimum")
> dev.off()
```

Exercise 14: Save the workspace (the current directory should be `swirl`) and the command history then quit R. Restart R in the `charlie` directory.

```
> library(limma)
```

Exercise 15: Read in the `quantarray` data in the `charlie` directory. Perform background correction on this data using the same three methods as above for the SPOT data. Make MA-plots of the data showing the effect of these background correction methods and save them to a file. Compare the MA-plots from the SPOT data in the `swirl` directory with the MA-plots of `Quantarray` data saved in the current (`charlie`) directory. Notice the characteristic *fanning* effect after subtracting background from `Quantarray` image output compared to the SPOT output above. The noise introduced at low intensities is typical of many image analysis programs which tend to over-estimate background measurements.

☞ Note that `read.maimages` might not work correctly for generic output files if you are working with an R console on a Mac.

```
> targets = readTargets("targetscharlie.txt")
> RG = read.maimages(targets$FileName, annotation = c("Block", "Row",
+ "Column", "Name", "ID"), columns = list(Rf = "F635 Mean", Gf = "F532 Mean",
+ Rb = "B635 Median", Gb = "B532 Median"))
```

```

> RG$targets = targets
> RG$printer = getLayout(RG$genes)
> dim(RG)
> RG.nb = backgroundCorrect(RG, method = "none")
> RG.sb = backgroundCorrect(RG, method = "subtract")
> RG.mb = backgroundCorrect(RG, method = "minimum")
> pdf("MAcharlie.pdf")
> par(mfrow = c(2, 2))
> plotMA(RG.nb, array = 2, xlim = c(0, 16), ylim = c(-7, 7), main = "bkgcorr: none")
> plotMA(RG.sb, array = 2, xlim = c(0, 16), ylim = c(-7, 7), main = "bkgcorr: subtract")
> plotMA(RG.mb, array = 2, xlim = c(0, 16), ylim = c(-7, 7), main = "bkgcorr: minimum")
> dev.off()

```

Exercise 16: Save the workspace (the current directory should be `charlie`) and the command history then quit R. Restart R in the `swirl` directory.

```
> library(limma)
```

Normalisation

i There are numerous normalisation methods available in `limma`. These include both within and between array methods for normalising log-ratios and also methods for the normalisation of the single-channel log-intensity data from two-colour experiments. In `limma` background correction may be performed during the normalisation step. This is especially useful if you already have a prior idea of which background correction method you want to use.

Exercise 17: Read the sections in the `limma` User's Guide on **Within-Array Normalization** and **Between-Array Normalization**. Now look up the help on `normalizeWithinArrays`; keep this help window open for reference during the following exercises.

🔖 Print-tip loess normalisation is the default normalisation method.

Exercise 18: Apply the normalisation method to the background corrected data from the `swirl` directory. Check the MA-plots (and spatial plots, if you have time) after this normalisation. Compare them to the MA-plots saved to file for this data. Make pin-group loess plots to assess the similarity in the loess fit for spots in the different grids on the array. Do you think pin-group loess normalisation is necessary or would a global loess fit be sufficient?

```

> MA = MA.RG(RG.mb)
> MAp = normalizeWithinArrays(RG.mb)
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+   plotMA(MAp, array = i)
+ }
> par(mfrow = c(1, 1))
> plotPrintTipLoess(MA, array = 1, layout = MA$printer)

```

```

> plotPrintTipLoess(MA, array = 2, layout = MA$printer)
> plotPrintTipLoess(MA, array = 3, layout = MA$printer)
> plotPrintTipLoess(MA, array = 4, layout = MA$printer)

```

Exercise 19: Perform global loess normalisation. Look at the file containing the MA-plots for these arrays, do you think global loess normalisation is required? Try global median normalisation also. Compare the MA-plots for these data after global loess and median normalisation.

```

> MA1 = normalizeWithinArrays(RG.mb, method = "loess")
> MAM = normalizeWithinArrays(RG.mb, method = "median")
> par(mfrow = c(2, 2), ask = T)
> for (i in 1:4) {
+   plotMA(MA, array = i, main = "no norm")
+   plotMA(MA1, array = i, main = "loess norm")
+   plotMA(MAM, array = i, main = "median norm")
+   plotMA(MAp, array = i, main = "pin-gp loess norm")
+ }
> par(ask = F)

```

Exercise 20: Use boxplots to compare the distribution of log-ratios between arrays after different normalisation methods. What does a boxplot summarise about a set of numbers (look up the help for `boxplot`)? Which normalisation methods affect the location (vertical position of the boxplot) of log-ratios and which also affect the overall variability of log-ratios within arrays (height of boxplots)?

```

> boxplot(MA$M ~ col(MA$M), main = "no norm")
> boxplot(MAM$M ~ col(MAM$M), main = "median norm")
> boxplot(MA1$M ~ col(MA1$M), main = "loess norm")
> boxplot(MAp$M ~ col(MAp$M), main = "pin-gp loess norm")

```

limma also has functions for performing between array normalisation using `normalizeBetweenArrays`. We will not be using this function in these practicals. However the issues about between array normalisation will be discussed in the lectures.

Exercise 21: (Optional) If time permits, you can apply similar normalisation techniques to the background corrected data from the `charlie` directory.