

limma: Basics

Natalie P. Thorne

November 18, 2008

Overview

i `limma` is a package for the analysis of gene expression microarray data, especially the use of linear models for analysing designed experiments and the assessment of differential expression. In this lab we will focus on the `limma` basics - reading in microarray image output files and associated information. Below is a brief overview of typical steps involved in getting started with a `limma` analysis of two colour microarray data sets.

♣ *Do not type in the following commands, they are there as a reference for the typical sequence of functions used to read data into `limma`. A typical preliminary analysis would begin by loading the `limma` library, reading in information about the target samples, reading in the image output files, identifying different spot types in the genelist, performing background correction and normalisation then making various plots.*

```
> library(limma)
> targets = readTargets()
> RG = read.maimages(targets$FileName)
> RG$targets = targets
> RG$printer = getLayout(RG$genes)
> types = readSpotTypes()
> RG$genes$Status = controlStatus(types, RG)
> RG = backgroundCorrect(RG)
> MA = normalizeWithinArrays(RG)
> plotMA(MA)
```

i `limma` is designed for analysing two-colour microarray data. However there are functions available for analysing Affymetrix and other single-colour array data. We will be concentrating on two-colour array data. For the analysis of two-colour microarray data, usually one image analysis output file is obtained for each array. Such files contain the various calculations on the foreground and background intensity measurements for each spot as well as other information like the Block, Row, Column, ID and probe names for each spot. In most cases it is also desirable to have a *targets* file which describes which RNA sample was hybridised to each channel of each array. A further optional file is the *spot types* file.

Targets file

☞ Read sections in the `limma` User's Guide on **Reading Two-Color Data**.

☞ Make sure the current directory for your R session is in the `limmaBasics` folder. All your analysis will happen in this directory. At any point during this session you may save the `workspace` so that you can come back to the same session later.

♣ *You can set the current directory by typing `setwd("/path/to/limmaBasics")`. Alternatively, in a GUI implementation (the point-and-click kind) use the option in the drop down menu to change to a different working directory.*

Exercise 1: Load the `limma` library and read in the targets file in the current directory. The targets file has been given the default name `Targets.txt`.

☞ Notice the use of the various arguments to `readTargets`; `path`, `sep` and `row.names`. Look at the help for this function to understand the default settings for these arguments.

```
> library(limma)
> targets = readTargets()
> targets
> targets = readTargets("Targets.txt")
> targets
```

☞ Although reading in the targets file can seem fairly easy, getting the targets file right can be difficult. In the next exercise I have deliberately created some of the common errors encountered when there are problems reading files into R, particularly the targets file.

☞ In many exercises in this and other labs you will be required to read in files from different directories. It is important to **remain in the current directory** during this R session (i.e. `limmaBasics` not `swirl` or `charlie`; don't change directory during this lab). To read in files in other directories, specify the *relative or full* directory path in the file name or set the `path` argument.

Exercise 2: Now read in the targets file located in the directory called `swirl`. Remember, don't change your working directory. Try reading in the comma delimited targets file called `TargetsCommas.txt`. In the same directory there are three incorrect targets files; `TargetsWrong1.txt`, `TargetsWrong2.txt` and `TargetsWrong3.txt`. For each of these, notice the errors given by R and make corrections to the files till you identify all the errors and are able to successfully read them into R.

♣ *Hint: for finding errors in a targets file, make sure there are no additional tabs (between columns or at the end of each row) or extra carriage returns at the end of the targets file. The best way to do this is to open the targets file in a text editor. Highlighting the text can be a useful way to find misplaced tabs at the end of rows or any extra carriage returns. Each row should contain the same number of columns, and should end with one carriage return (including the last row of the file). Unusual characters such as the # symbol can cause problems for R when reading in files. Note that opening the files in Excel will not help you to find hidden extra tabs or carriage returns.*

```

> targets = readTargets(path = "swirl")
> targets = readTargets("TargetsCommas.txt", path = "swirl", sep = ",")
> targets = readTargets("TargetsWrong1.txt", path = "swirl")
> targets = readTargets("TargetsWrong2.txt", path = "swirl")
> targets = readTargets("TargetsWrong3.txt", path = "swirl")

```

Reading in Data

❏ `limma` allows you to easily read in microarray data from multiple arrays within an experiment. It uses a function called `read.maimages` to read in image output files one at a time. It extracts the columns of interest from each file, sequentially building up an `RGList` object containing the necessary information from each array.

Exercise 3: Read the section in the `limma` User's Guide on **Reading in Intensity Data**.

❏ `limma` supports reading in microarray data files from various image analysis programs such as Agilent, Array Vision, BlueFuse, ImaGene, GenePix, QuantArray, ScanArray Express, SMD and SPOT.

Exercise 4: Read the swirl data into R using the `read.maimages` function. Look up the help on this function and make sure you are familiar with the required arguments. This function may take a while to run, so be patient. Again, don't change directories, instead specify the `path` argument.

☞ The swirl data set consists of two sets of dye-swap experiments making a total of four replicate hybridizations. Each of the arrays compares RNA from swirl fish with RNA from normal ("wild type") fish.

♣ *If you get an error saying "cannot open file", go back and check the targets file. There may still be errors in the targets file (that did not prevent it being read in) such as incorrect filenames for the image analysis output files.*

☞ `RG` is a list object. It contains named objects, `R`, `G`, `Rb` and `Gb`, matrices for the red and green foreground and background spot intensities respectively. Rows in these matrices correspond to genes, and columns correspond to arrays.

☞ The swirl arrays were image analysed using Spot. This is the default for the `source` argument in `read.maimages`. For completeness in the exercise below we specify the `source` argument anyway. What objects are read in by `read.maimages` from SPOT output files? What are the array names (i.e. column names of data matrices in `RG`)?

```

> RG = read.maimages(targets$FileName, path = "swirl", source = "spot")
> show(RG)
> names(RG)
> colnames(RG$R)

```

❏ The list returned by `read.maimages` is a special kind of list. It is a kind of list object that is recognised in a special way by the `limma` library functions. It is called an `RGList`

not just a `list`. There are also `MAList` objects in `limma` too. Many of `limma`'s functions expect an `RGList` or `MAList` object. It makes writing functions easier, because you can give one of these objects as an argument to a `limma` function and it will automatically know what and where the information is stored in these objects. In this way, fewer arguments need to be specified in a function because the `limma` function knows exactly what to extract from the list. The function `plotMA` is a good example of a `limma` function which recognises the `RGList` or `MAList` object types. You will become very familiar with this function later on.

☞ In `plotMA` you don't need to specify the `x` and `y` axis data, you can give it an `RGList` object, it will calculate `M` and `A` values and by default plot `M` versus `A` for the first array. There are more fancy capabilities to `plotMA` that you will see later.

```
> is.list(RG)
> is(RG, "RGList")
> class(RG)
> plotMA(RG)
```

Exercise 5: Practise subsetting the `RGList` object which we named `RG`. Make sure you understand what sets of data you are getting from each command below. Calculate the signal to noise ratio for the red channel for the second array.

```
> RG$R[1:5, ]
> RG$Gb[1001, ]
> s2n = RG$R[, 2]/RG$Rb[, 2]
> plot(s2n)
> boxplot(s2n)
```

Saving and recovering the workspace

Exercise 6: Save the current workspace and `Rhistory` and close down `R`. Now restart `R` in the same directory and load this workspace and command history again. Use `objects()` to see that you have recovered all the analysis you performed.

☞ In the GUI version of `R` you can use the pull down menus to save the current workspace and then load it again. It will save to the current directory - so be careful to note what directory you are in (should be `limmaBasics`). After you restart `R`, make sure you are in the correct directory before loading the workspace. The workspace gets saved as a hidden file. By default this hidden file is called `.RData` - in Windows this will appear as a large semi-transparent blue `R` icon; in Mac's you won't see any icon when viewing files in a directory - but it will be there! Alternatively, you can name the workspace, for example `myRanalysis.RData`.

☞ To save the history of your commands, use the drop down menu item `save history`. This history will be saved to the current directory also, with a default name that `R` will recognise and restore when the workspace is started again in the same directory.

```

> save.image()
> save.image("myRanalysis.RData")
> savehistory()
> savehistory("myHistory.Rhistory")
> q()
> load("myRanalysis.RData")
> loadhistory("myHistory.Rhistory")
> objects()

```

Exercise 7: Make sure you can access the objects you had created before you quit R. Can you see the recent history of commands from the R session before you quit? Check that the current working directory is again `limmaBasics`.

♣ *Each time you restart R you will also have to reload the `limma` library.*

Gene annotation

📖 For most image analysis programs, the basic gene annotation information is contained within each image output file. Some formats do not contain the gene annotation information and some formats have column headings for the annotation information that are different from the default settings in `limma`.

Exercise 8: Read the sections in the `limma` User's Guide on **Reading the Gene list**. Get the gene annotation information for the swirl `Spot` arrays and store it in the `genes` component of `RG`.

♣ *Spot image analysis program is unusual because it does not contain gene annotation information in the output files. Therefore, for `Spot` files, one must read in the gene annotation information from a separate file called the `GAL` file.*

```

> library(limma)
> RG$genes = readGAL(path = "swirl")
> names(RG)
> dim(RG$genes)
> colnames(RG$genes)
> RG$genes[1:10, ]

```

Exercise 9: Get the print grid layout for the arrays in this data set. Confirm for yourself that the printer layout obtained is correct and consistent with the number of genes (rows of `RG`). Take a look at the objects stored in `RG` now. Make a new `RGList` object for the first two arrays only.

🔗 The layout information which gets stored in the `printer` component will be used later. For example, the layout information is required for making spatial plots and for pin group (block or grid) normalisation.

☞ You might notice that, unlike ordinary `list` objects the `RGList` and `MAList` class of objects in `limma` have a special subset ability. For example, `dim` works on an `RGList` but not on ordinary lists in R. Below we subset the first two arrays only and create a new `RGList` object called `RG12`.

```
> RG$printer = getLayout(RG$genes)
> RG$printer
> is(RG, "RGList")
> class(RG)
> names(RG)
> dim(RG)
> RG12 = RG[, 1:2]
> is(RG12, "RGList")
> class(RG12)
> show(RG12)
```

♣ *We now turn to a another example data set to practise reading in microarray data.*

Exercise 10: There is no targets file for the charlie array data. Make your own targets file for the slides in the directory called `charlie`. This targets file will contain the `FileName` of the output files in the folder `charlie`. Put this file in the `charlie` directory and make sure you can read in the targets file you create. Read in the data without changing the current working directory. Check that the gene annotation information gets stored and determine the printer layout of the arrays.

♣ *You might find it easiest to prepare the targets file in Excel, but remember to save it as a .txt file (call it `charlietargets.txt`). Use the targets file for the swirl data as a template.*

☞ Each array compares a treated cell line with a wild type cell line. Slides 534 and 535 have the treated samples in the red (Cy5) channel where as slides 532 and 533 have the treated samples in the green (Cy3) channel (i.e. they are dye-swaps). All arrays were hybridised on the 28th of August 2004. The image analysis output files have the extension `.out`.

☞ The output files in `charlie` are not a specific format of any image analysis program supported by `limma`. Read the help page on `read.maimages` to see how to specify which columns of the output file to read into the `RGList` object. You will also notice that you will need to specify the `annotation` argument because `read.maimages` assumes they may be non-standard column names because you had to specify the column names for the foreground and background intensity data.

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

```
> targets.charlie = readTargets("charlietargets.txt", path = "charlie")
> RG.charlie = read.maimages(targets.charlie$FileName, columns = list(Rf = "F635 Mean",
+   Gf = "F532 Mean", Rb = "B635 Median", Gb = "B532 Median"), path = "charlie")
> names(RG.charlie)
```

```

> RG.charlie = read.maimages(targets.charlie$FileName, annotation = c("Block",
+   "Row", "Column", "Name", "ID"), path = "charlie", columns = list(Rf = "F635 Mean",
+   Gf = "F532 Mean", Rb = "B635 Median", Gb = "B532 Median"))
> names(RG.charlie)
> RG.charlie$targets = targets.charlie
> RG.charlie$printer = getLayout(RG.charlie$genes)
> names(RG.charlie)
> RG.charlie$printer
> dim(RG.charlie)

```

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

Spot types file

📖 Spot types files contain information about the various types of spots that are on your array. Like the targets file, this is another tab-delimited text file. It allows you to identify different types of spots from your genelist, control their plotting character, colour and other plotting parameters in `plotMA`.

Exercise 11: Read in the spot types file that has been created for the swirl data. Then use the function `controlStatus` to set the control status for each spot on the array.

☞ In the spot types file for swirl, you are asking it to find two types of spots. The first (and most general) is a spot which has been defined as `SpotType gene`. The `SpotType` name is the user defined name. The `ID` and `Name` columns are determined by the entries in the `genes` matrix. The second type of spot defined in this `SpotType` file is the `control` type. These are also a type called a `gene`, but are more specific than a `gene` (which can be any spot on your genelist). `SpotTypes` called `control` have "control" as their `ID` for these arrays.

```

> types = readSpotTypes(path = "swirl")
> types
> Status = controlStatus(types, RG)
> plotMA(RG, status = Status)

```

Exercise 12: Read in the second `SpotTypes` file in the `swirl` directory called `SpotTypes2.txt`. Fix the error in the file so that you can successfully read in the `SpotTypes` file. Make a new control status and make a new `plotMA`. Then try changing entries in the file yourself to adjust the appearance of the control spots in the `MA`-plot.

```

> types2 = readSpotTypes("SpotTypes2.txt", path = "swirl")
> Status2 = controlStatus(types2, RG)
> plotMA(RG, status = Status2)

```

☞ Before attempting the next exercise, read the section in the `limma` User's Guide on **The Spot Types File**.

Exercise 13: Search through the various `Name`'s of the control genes and write a new `SpotTypes` file which allows you to identify some different types of controls on the slide. Look in the file called `SpotTypes3.txt` where this has been started for you. Make sure you understand the concept of pattern matching for identifying spots of different kinds in the `SpotTypes` file.

☞ Notice that `SpotTypes3.txt` currently has most of the technical controls for the arrays specified. The remaining controls in these arrays are most likely biological controls.

☞ Use the first line of code below to see a list of the different `Name`'s of genes with `ID` equal to `control`. Look in `SpotTypes3.txt` and add at least one of these control genes that is not already specified in this `SpotTypes` file.

```
> unique(RG$genes$Name[RG$genes$ID == "control"])
> types3 = readSpotTypes("SpotTypes3.txt", path = "swirl")
> Status3 = controlStatus(types3, RG)
> plotMA(RG, status = Status3)
```