R/MAANOVA: An extensive R environment for the Analysis of Microarray Experiments

Hao Wu, Gary A. Churchill

May 9, 2006

Contents

1 Introduction

R/maanova is an extensible, interactive environment for the analysis of microarray experiments. It is implemented as an add-on package for the freely available statistical language R (www.r-project.org). The engine functions were written in C for better performance.

MAANOVA stands for MicroArray ANalysis Of VAriance. It provides a complete work flow for microarray data analysis including:

- Data quality checks and visualization

- Data transformation
- ANOVA model fitting for both fixed and mixed effects models
- Statistical tests including permutation
- Cluster analysis with bootstrapping

R/maanova can be applied to any microarray data but it is specially tailored for multiple factor experimental designs. Mixed effects models are implemented to estimate variance components and perform F and T tests for differential expressions.

2 Installation

2.1 System requirements

This package was developed under *R 1.8.0* in the *Linux Redhat 8.0* operating system. The programs have been observed to work under *Windows NT/98/2000*. The memory requirement depends on the size of the input data but a minimum of 256Mb memory is recommended.

2.2 Obtaining R

R is available in the Comprehensive R Archive Network (CRAN). Visit `http://cran.r-project.org` or a local mirror site. Source code is available for UNIX/LINUX, and binaries are available for Windows, MacOS, and many versions of Linux.

2.3 Installation - Windows(9x/NT/2000)

- Install *R/maanova* from Rgui

  1. Start Rgui
  2. Select Menu `Packages`, click `Install package from local zip file`. Choose the file `maanova_*.tar.gz` and click 'OK'.

- Install *R/maanova* outside Rgui

  1. Unzip the `maanova_*.tar.gz` file into the directory `$RHOME/library` (`$RHOME` is something like `c:/Program Files/R/rw1081`). Note that this should create a directory `$RHOME/library/maanova` containing the R source code and the compiled dlls.
  2. Start Rgui.
  3. Type `link.html.help()` to get the help files for the maanova package added to the help indices.

2.4 Installation - Linux/Unix

1. Go into the directory containing `maanova_*.tar.gz`.
2. Type `R CMD INSTALL maanova` to have the package installed in the standard location such like `/usr/lib/R/library`. You will have to be the superuser to do this. As a normal user, you can install the package in your own local directory. To do this, type `R CMD INSTALL -library=$LOCALRLIB maanova_*.tar.gz`, where

`$LOCALRLIB` is something like `/home/user/Rlib/`. Then you will need to create a file `.Renviron` in your home directory to contain the line `R_LIBS=/home/user/Rlib` so that R will know to search for packages in that directory.

3 Data and function list

The following is a list of the available functions. For more information about the function usage use the online help by typing "`?functionname`" in *R* environment or typing `help.start()` to get the html help in a web browser.

1. Sample data available with the package

   `abf1` A 18-array affymetric experiment. 500 hand picked genes are included in the data set

   `kidney` A 6-array kidney data set from CAMDA (Critical Assessment of Microarray Data Analysis).

   `paigen` A multiple factor 28-array experiment from Bev Paigen's lab in The Jackson Lab. Only 300 hand picked genes are included in this data set

2. File I/O

   `read.madata` Read microarray data from TAB delimited simplex text file.

   `write.madata` Write microarray data to a TAB delimited simple text file.

3. Data quality check

   `arrayview` View the layout of the arrays

   `riplot` Ratio intensity plot for arrays

   `gridcheck` Plot grid-by-grid data comparison for arrays

   `dyeswapfilter` Flag the bad spots in dye swap experiment Note that these data quality check functions only work for 2-dye arrays at this time.

4. Data transformation

   `createData` Create a data object with options to collapse the replicated spots and do log2 transformation

   `transform.madata` Data transformation with options to use any of several methods

5. ANOVA model fitting

   `makeModel` Make model object to represent the experimental design

   `fitmaanova` Fit ANOVA model

   `resiplot` Residual plot on a given ANOVA model

6. Hypothesis testing

`matest` F-test or T-test with permutation

`adjPval` Calculate FDR adjusted P values given the result of [`matest`]

`volcano` Volcano plot for summarizing F or T test results

7. Clustering

`macluster` Bootstrap clustering.

`consensus` Build consensus tree out of bootstrap cluster result

`fom` Use figure of merit to determine the number of groups in K-means cluster

`geneprofile` Plot the estimated relative expression for a given list of genes

8. Utility functions

`fill.missing` Fill in missing data.

`summary.madata` Summarize the data object.

`summary.mamodel` Summarize the model object.

`subset.madata` Subsetting the data objects.

`exprSet2Rawdata` Convert an object of exprSet to an object of Rawdata.

4 Preparing the input files

Before using the package, the user must manually prepare a data file and a design file.

4.1 Preparing the data file

There is only ONE data file for all of the slides in an experiment. Most gridding software produces one file for each slide. Thus you will have multiple files for a multiple array experiment and you have to combine these files to one data file as the input for *R/maanova*.

The data file is a TAB delimited text file. Each rows corresponds to the data for a gene. In the first a few columns, you can put some gene information, e.g., the Clone ID, Gene Bank ID, etc. and the grid location of the spot. Note that some gridding softwares return Block numbers instead of metarow and metacol. Then you must manually compute metarow and metacol from block and put them in the file. After that you need to put the scanned data for all arrays in the rest of the columns. (You need to make the decision what data you want to use in analysis, e.g., mean versus median, background subtracted or not, etc.) For N-dye arrays, the N channel intensity data for one array need to be adjacent to each other (in consecutive columns). The order (dye1, dye2, ...) must be consistent across all of the slides. You can put the spot flag as a column after intensity data for each array. (Note that if you have flag, you will have N+1 columns of data for each array. Again, this must be consistent for all arrays in the data set.) If you have duplicated spots within one array, replicated measurements of the same clone on the same array should appear in adjacent rows. This can be easily done by sorting on cloneid. The number of replicates must be constant for all genes.

As an example, you have four slides for a 2-dye array experiment scanned by GenePix. Then you will have four output files. Following the steps to create your data file:

1. Open your favorite spread sheet editor, e.g., MS Excel and create a new file;
2. Paste your clone ID, Gene names, Cluster ID and whatever information you want to keep into the first several columns;
3. Open your first GenePix file in another window, copy the grid location into next 4 columns (you only need to do this once because they are all the same for four slides);
4. Copy the two columns of foreground mean value (if you want to use it) and one column of flag to the file in the order of Cy5, Cy3, flag;
5. Open your other 3 files and repeat step 4;
6. Select the whole file and row sort it according to Clone ID;
7. Save the file as tab delimited text file and you are done.

The data file must be "full", that is, all rows have to have same number of columns. Sometimes leading and trailing TAB in the text file can cause problems, depending on the operating system. So the user should be careful about that. Sometimes the special characters in gene description can cause reading problem. I don't encourage you to put the gene description in the data file. If you have to do that, you must be careful (sometimes you need to remove the special characters manually).

4.2 Preparing the design file

The design file is another TAB delimited text file. The number of rows in this file equals the number of arrays times N(the number of dyes) plus one (for the column headers). The number of columns in this file depends on the experimental design. For example, you can have "Strain", "Diet", "Sex", etc. in your design file. You must have the following columns in the design file (column headers are case sensitive):

- Array: for array name
- Dye: for dye name
- Sample: Sample ID number

Array and Dye columns are easy to understand. Sample column contains integers used to identify biological replicates and reference samples. Usually you should assign each biological individual a unique Sample number. Reference samples are represented by zero(0). Reference sample are treated differently. They will always be treated as fixed factor in the model and not involved in statistical tests.

You must not have the following columns in the design file:

- Spot: reserved for spot effect
- Label: reserved for labelling effect

Sample column in design file need to be continuous integer. All other columns can be either integers or characters.

You don't have to USE all factors in design file. When making the model object in `makeModel`, the experimental design will be determined by the design and a formula. You can put all factors in design file but turn them on/off in formula.

5 A quick tour of the functions

This section will go through a few demo scripts distributed with the package to help the users understand the function syntax and capabilities. The binary data are distributed with the software. The original data file can be downloaded from:
`http://www.jax.org/staff/churchill/labsite/software/`

5.1 CAMDA kidney experiment

This is the kidney data from CAMDA (Critical Assessment of Microarray Data Analysis) originally described by Pritchard *et al.*

The website for CAMDA is `http://www.camda.duke.edu`. It is a 24-array double reference design. Six samples are compared to a reference with dye swapped and all arrays are duplicated. Flag for bad spots is included in the data.

Note that because BioConductor requires all contributed packages to be less than 1Mb in size, the binary data distributed with the package is only a small part of the real data set (first 300 genes). So you cannot reproduce the figures presented in following sections. The package with full data set, can be download from Gary Churchill's website at `http://www.jax.org/staff/churchill/labsite/software/`. You can also find the original data file (in text format) there.

1. First load data into the workspace

   ```
   R> data(kidney)
   ```

2. Then we do some data quality check

   ```
   R> gridcheck(kidney.raw)
   ```

   `gridcheck` is used to check the hybridization and gridding quality within and cross arrays. You should see near linear scatter plots in all grid for all arrays. This one is not great but acceptable. The grid check plot for the first array is shown in figure 1. The red dots are for the spots with flags.

   ```
   R> riplot(kidney.raw)
   ```

   `riplot` stands for ratio-intensity plot. It is also called MA plot. The riplot for the first array is shown in figure 2.

   ```
   R> arrayview(kidney.raw)
   ```

   `arrayview` is used to view the spatial pattern of the arrays. The standardized log ratios for all spots are shown in different colors. The arrayview for the first array is shown in figure 3.

   You will generate a lot of figures by doing gridcheck, riplot and arrayview. Use `graphics.off` to close all figures.

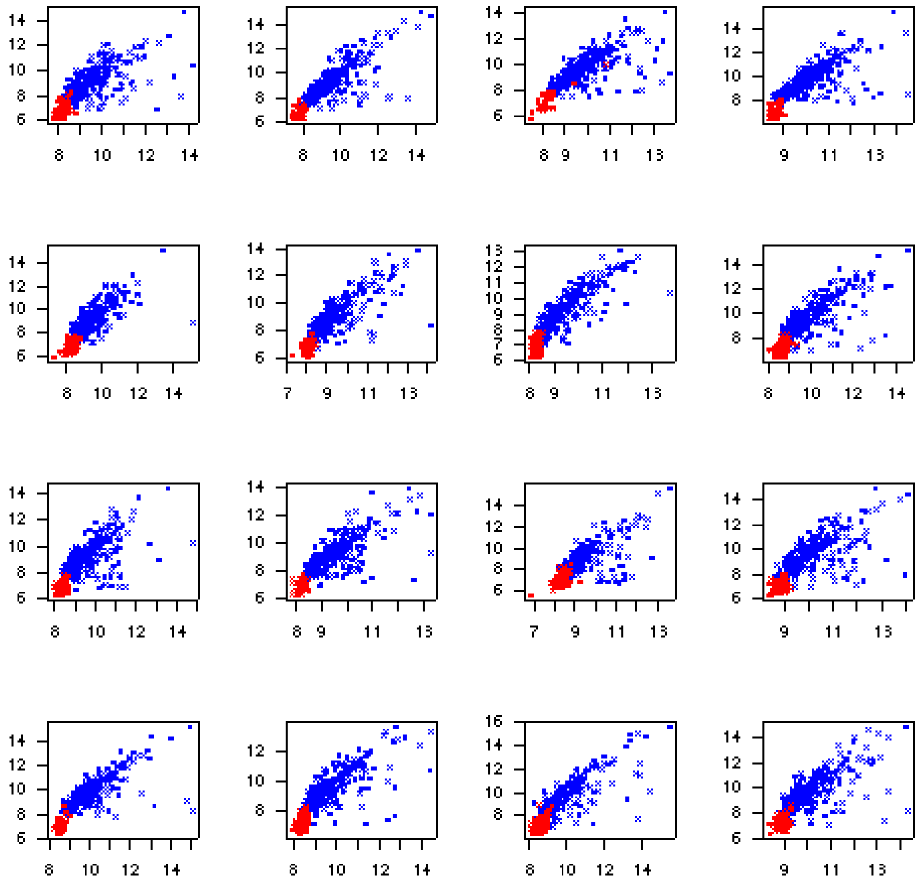

Figure 1: Grid check plot for the first array in kidney data

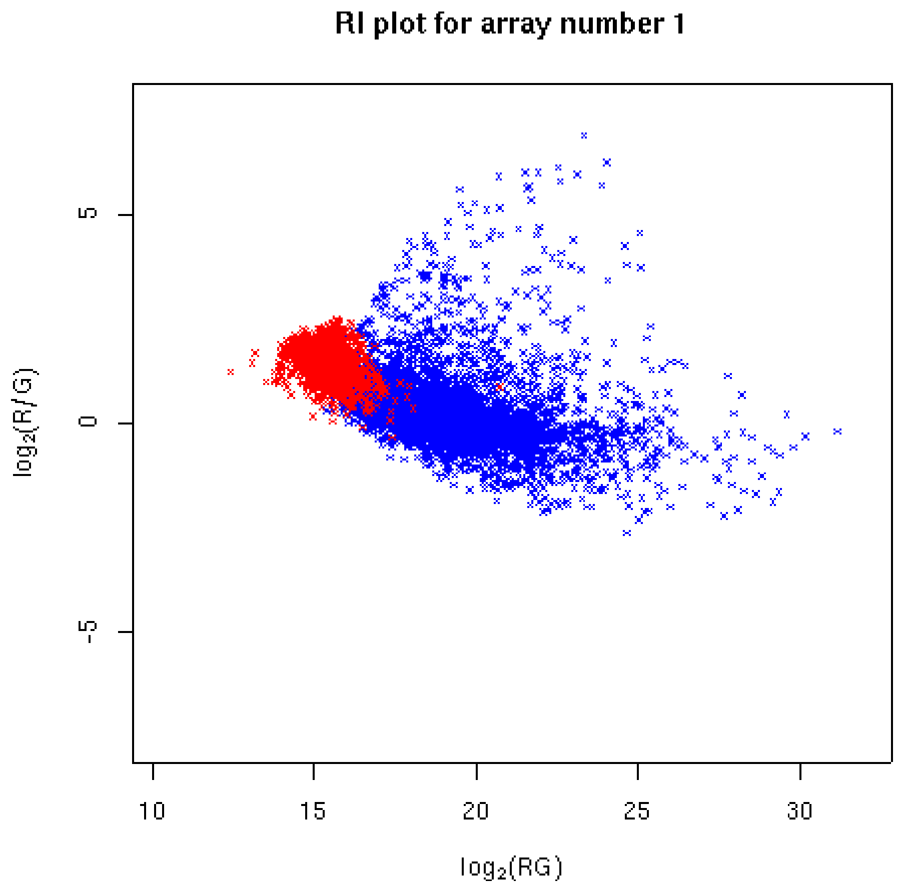

Figure 2: RI plot for the first array in kidney data

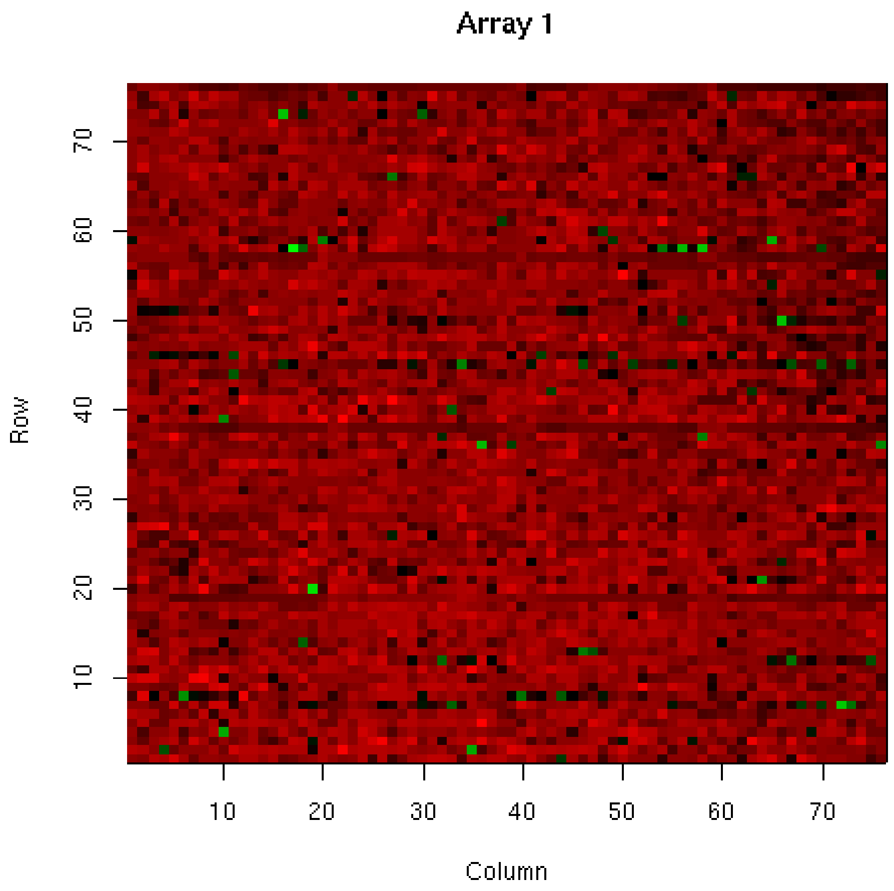

Figure 3: arrayview for the first array in kidney data

3. Now make an object of class `madata`. `madata` and `mamodel` are the key objects in *R/maanova*. Most of the functions work on them. A `madata` object stores the experimental data. It derived from the raw data. The `mamodel` object stores the experimental design information. We will discuss it a little later.

   ```
   R> kidney <- createData(kidney.raw)
   R> summary(kidney)
   ```

4. Transform the data using spatial-intensity joint loess.

   ```
   R> kidney <- transform.madata(kidney, method="rlowess")
   ```

   There are several data transformation method. Which method to use depends on the data. Read Cui *et al.*(2002) for detail. The transformation plot for the first array is shown in figure 4.

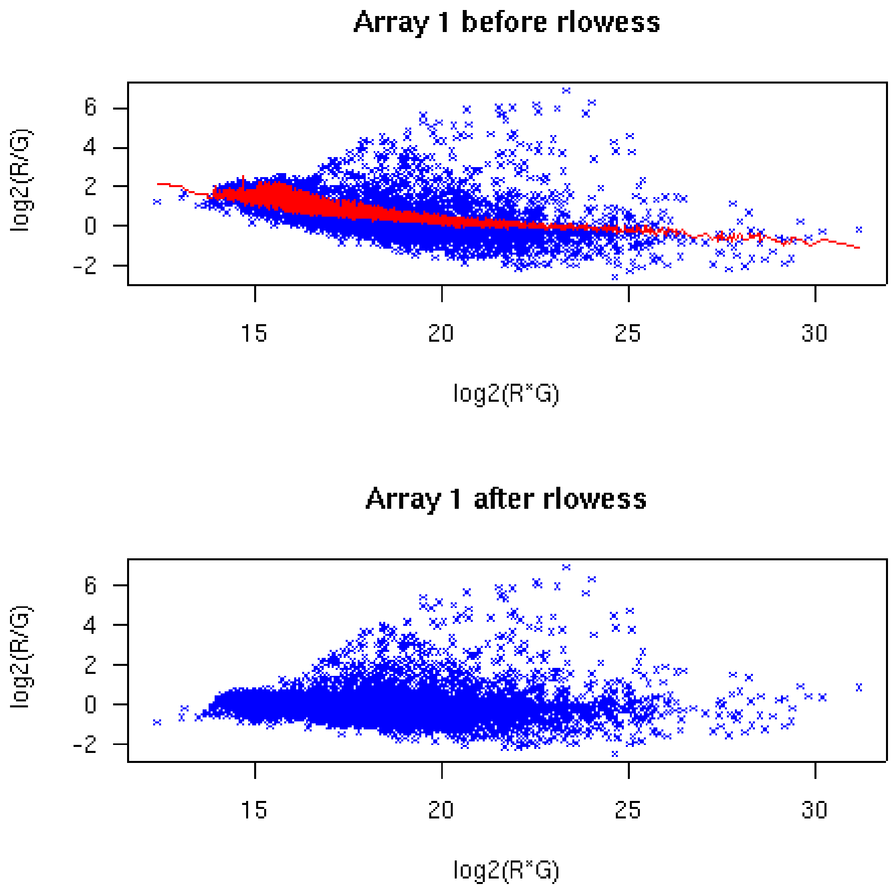

Figure 4: Joint lowess transformation on the first array for the kidney data

5. Make model object for fixed model

   ```
   R> model.fix <- makeModel(data=kidney, formula=~Dye+Array+Sample)
   R> summary(model.fix)
   ```

   A `mamodel` object store the experimental design information. `makeModel` function takes a data object and a R formula as the ANOVA model and make the design matrices. The input formula is an object of `formula`. It represents the ANOVA model. In the formula, you can put any combination of the factors in your design. Interaction between any two terms are allowed. At this point, 3 or higher way interactions are not taken by the program. `makeModel` takes another argument `random` for the random terms. `random` must be another formula. All terms in `random` must be included in `formula` as well.

6. Fit ANOVA model and do residual plot

   ```
   R> anova.fix <- fitmaanova(kidney, model.fix)
   R> resiplot(kidney, anova.fix)
   ```

7. Permutation F test. F-test function can test one or multiple terms in a given model object. User can do residual shuffling or sample shuffling for fixed effect models. For mixed effects models, only sample shuffling is available. Permutation test could be very time consuming, especially for mixed effects models. The permutation test function can run on linux clusters through message passing interface (MPI). For detailed information about F test and using computer cluster, please read appendix.

   Here we want to test Sample effect in the model:

   ```
   R> test.fix <- matest(kidney, model=model.fix, term="Sample",
          n.perm=500)
   ```

   Now `test.fix` contains the tabulated P values and permutaion P values. Sometimes we want to calculate FDR adjusted P values for the test result.

   ```
   R> test.fix <- adjPval(test.fix)
   ```

   After getting the F-test result, We can do volcano plot to visualize it. `volcano` function has options to choose the P-values to use and set up thresholds. We will use tabulated P values for F1 and FDR adjusted permutation P values for other tests here. In the plot, the orange dots above the horizontal line represent the significant genes. Note that the the flagged spot can be highlighted in the plot. You can turn if on by providing `highlight.flag=TRUE`.

   ```
   R> idx.fix <- volcano(test.fix,method=c("unadj", rep("fdrperm",3)),
            highlight.flag=FALSE)
   ```

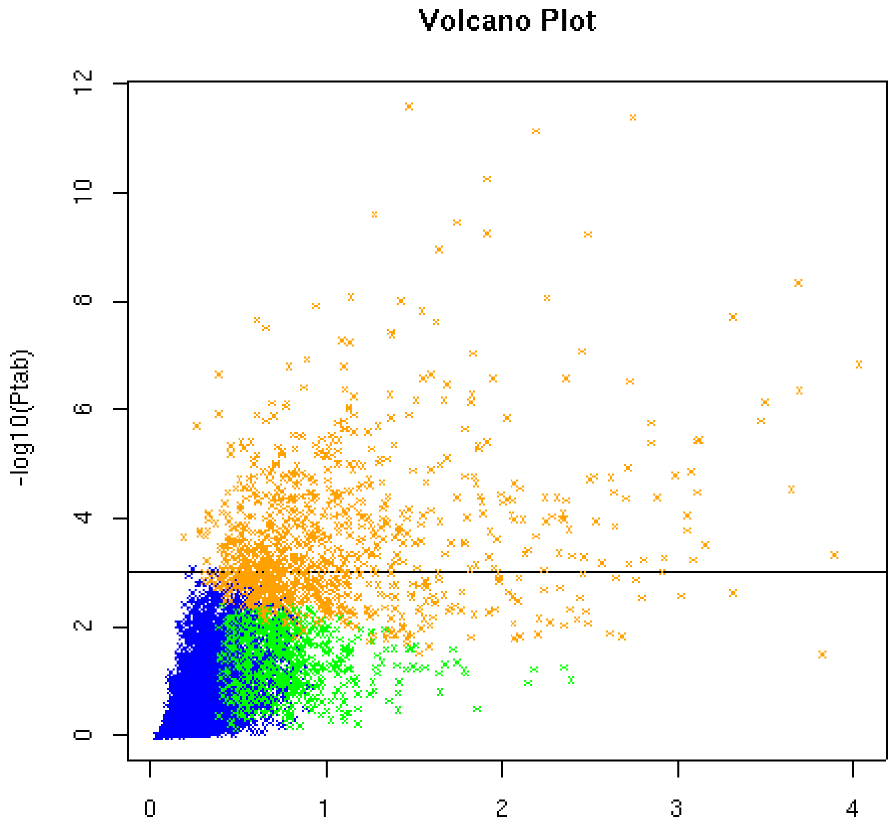

Figure 5: Volcano plot for kidney data - fixed effect model

The volcano plot is shown in figure 5. Note that the return variable of volcano contains the indices for significant genes.

8. Now we can do cluster bootstrapping and build consensus trees. Currently two cluster methods are implemented, hierarchical clustering and K-means. Hierarchical cluster could be very sensitive to bootstrap if you have too many leaves on the cluster. Some small disturbance on the data could change the whole tree structure. So if you have many genes, say, more than 50, and you want to build a consensus tree from 100 bootstrapped hierarchical trees. It is very likely that you get a comb, that is, all leaves are directly under root. But if you only have a few genes to cluster, it is working fine. So I suggest user use K-means to cluster the genes and use hierarchical to cluster the samples.

   `macluster` is the function to do cluster bootstrapping and `consensus` is used to build consensus trees (groups for K-means) from the bootstrap results.

   ```
   R> cluster.kmean <- macluster(anova.fix, ,term="Sample",
            idx.gene=idx.fix$idx.all,what="gene", method="kmean"
            kmean.ngroups=5, n.perm=100)
   R> con.kmean <- consensus(cluster.kmean, 0.7)
   ```

   An expression profile plot will be generated for the consensus K-means result. The plot is shown in figure 6.

   Now we can do hierarchical cluster on the samples. The consensus tree is shown in figure 7.

   ```
   R> cluster.hc <- macluster(anova.fix, term="Sample",
         idx.gene=idx.fix$idx.all,what="sample", method="hc", n.perm=100)
   R> con.hc <- consensus(cluster.hc)
   ```

9. Now we are going to analyze the data using mixed effect model. First we make a model object with Array effect as random factor. Note that normally Array effect, Spot effect and Labeling effect should be treated as random. Since we don't have technical replicate here, Spot and Label cannot be fitted.

   ```
   R> model.mix <- makeModel(data=kidney, formula=~Dye+Array+Sample,
              random=~Array)
   R> summary(model.mix)
   ```

10. Then we can fit the ANOVA model. This will take quite a while to finish. EM algorithm is implemented in the engine function for solving MME. For details about MME and the EM algorithms, read Searle *et al.*.

    ```
    R> anova.mix <- fitmaanova(kidney, model.mix)
    ```

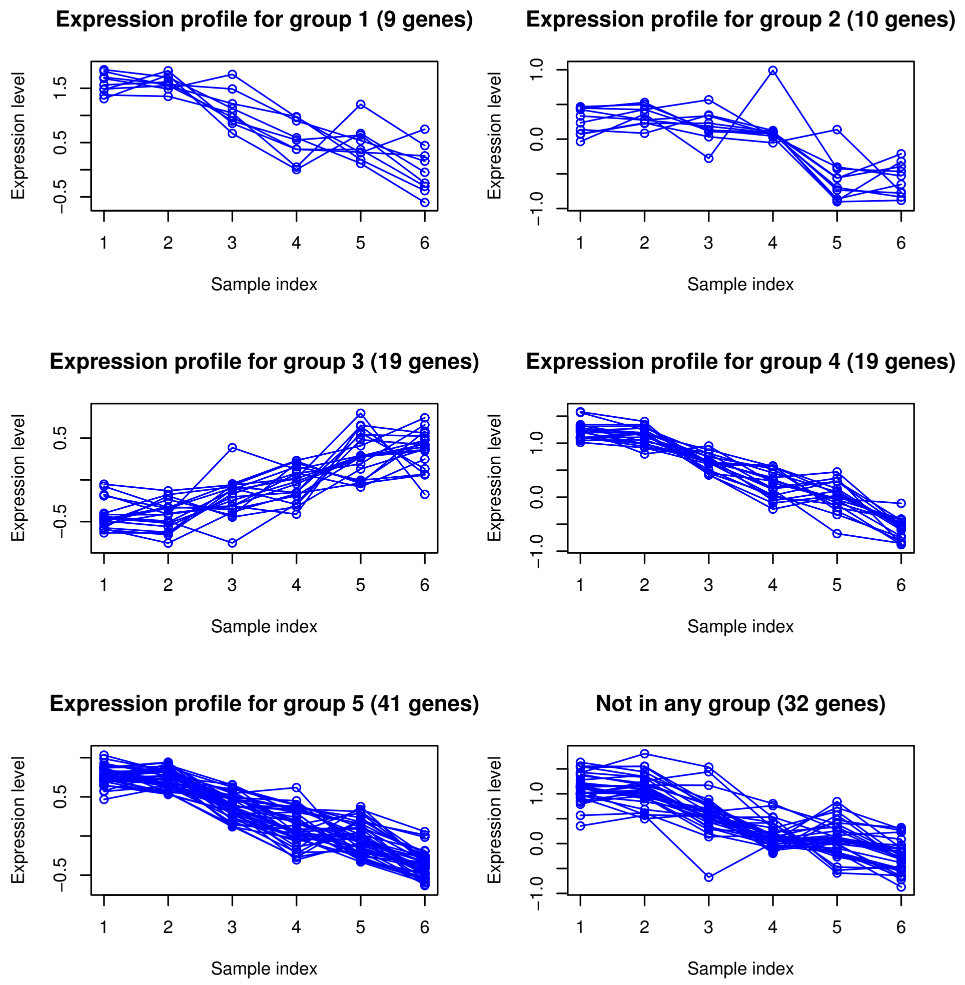

Figure 6: Expression profile plot for bootstrap K-means result, kidney data

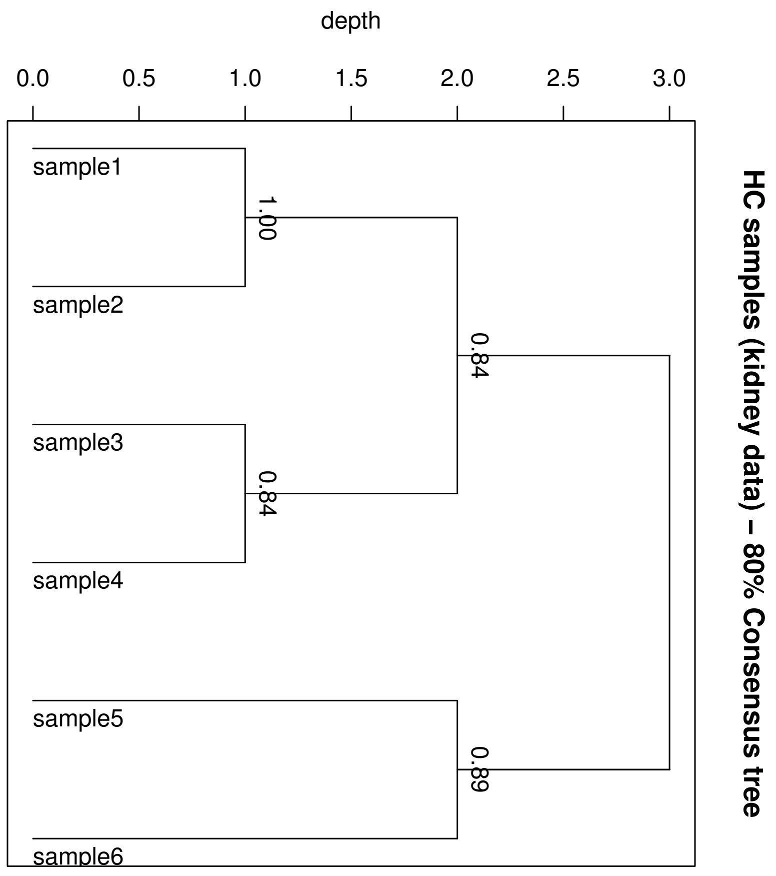

Figure 7: 80% Consensus tree for bootstrapping hierarchical cluster on the samples, kidney data

11. Now we test the sample effect. Again, permutation test for mixed effects model is very slow. You have better run it on computer clusters if possible.

    ```
    R> ftest.mix <- matest(data=kidney, model=model.mix, term="Sample",
           n.perm=100)
    ```

    We can do volcano plot for the mixed model result.

    ```
    R> idx.mix <- volcano(anova.mix, ftest.mix)
    ```

    The rest of the analysis (clustering and consensus trees) are skipped here.