The genArise Package

Ana Patricia Gómez Mayén
Gustavo Corral Guillé
Lina Riego Ruiz
Gerardo Coello Coutiño

October 16, 2005

1 Introduction

genArise is a package that contains specific functions to perform an analysis of microarray obtained data to select genes that are significantly differentially expressed between classes of samples. Before this analysis, genArise carry out a number of transformations on the data to eliminate low-quality measurements and to adjust the measured intensities to facilitate comparisons.

First, you must install the R system. The current released version is 2.0.1 and the binary can be obtained via CRAN, a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries. The CRAN master site can be found at the URL `http://cran.R-project.org/`.

Then, you need install the packages *tkrplot*, *locfit* y *xtable* in order to be able to use this package. This can be done as follows:

```
> install.packages(c("locfit","tkrplot", "xtable"))
```

This R function automatically download and install the three packages from CRAN, but you must be connected to the network. In other case, you must download the three packages from CRAN for the correct O.S. (Windows, Linux) and install from the terminal these add-on packages with the

script *R CMD INSTALL package_name* for Linux or *Rcmd INSTALL package_name* for Windows. With the same script you can install genArise too. You can read the instalation manual in the URL: `http://www.ifc.unam.mx/genarise`.

genArise GUI uses widgets from the BWidget package and from the Img package, neither of which come bundled with the default installation of R. If you are using Windows (and not using the installation wizard), you need to install ActiveTcl 8.4.x from `http://aspn.activestate.com/ASPN/Downloads/ActiveTcl/`. After install ActiveTcl, you must set the environment variable MY_TCLTK to a non-empty value, for example: `MY_TCLTK=Yes` and it is assumed that you want to use a different Tcl/Tk 8.4.x installation, and that this is set up correctly. You must too set the environment variable TCL_LIBRARY with the path where are the DLLs of Tcl in your machine, for example: `TCL_LIBRARY=C:/Tcl/lib/tcl8.4`.

If using Linux, install the Tcl and Tk 8.3 rpms from CD, e.g. into /usr/lib, then download BWidget from `http://sourceforge.net/projects/tcllib/`. You must gunzip and untar this file into /usr/lib. Download too tkimg1.3 (this is the version for Tcl and Tk 8.3) which you can search in `http://sourceforge.net` and gunzip it and untar it into /usr/lib. Then you probably must copy the directory /usr/lib/tkimg1.3/Img/exec_prefix/lib/Img/ to /usr/lib.

Finally, genArise can be loaded with the next instruction:

```
> library(genArise)

Loading required package: locfit
Loading required package: lattice
Locfit for R.
August 3, 2000.

Attaching package: 'locfit'


        The following object(s) are masked from package:stats :
```

```
        knots

Loading required package: tkrplot
Loading required package: tcltk
Loading Tcl/Tk interface ... done
Loading required package: xtable

Attaching package: 'genArise'


        The following object(s) are masked from package:utils :

        help
```

Once loaded, you can use any function of genArise and proceed with the analysis of the microarray data. genArise is provided with functions that can be applied from R prompt in every step of analysis; however, there is also a graphical user interface to facilitate the use of all the functions in the package.

2 The genArise Package's Functions

To start the analysis you must read first the data contained in the input file. To read this file, genArise provide the function `read.spot`. This function returns an object of type Spot (a table structure with the name of the file without extension and the data for the analysis: Cy3 intensity, Cy5 intensity, Cy3 Background, Cy5 Background, Ids) because many other functions of genArise that carry out transformations on the data need a spot object as argument. The syntax for this function is:

```
> my.spot <- read.spot( file.name, cy3 = 1, cy5 = 2,
  bg.cy3 = 3, bg.cy5 = 4, ids = 5, header = FALSE,
  sep = "\t", is.ifc = FALSE)
```

```
Output: an object of class Spot called my.spot
```

You can see the details and the meaning of each one of the arguments of this and all the other functions described bellow in the manual genArise.pdf

and you can download it from this address `http://www.ifc.unam.mx/genarise` in the `Package GenArise` *API* link

We are not to give an example from this function, but you just have to follow the sintax of the `read.spot` function described in the manual to create a spot object. In this manual we use a subset of a data set called Simon for the next examples. Simon, as we have said, is a spot object that was created with the `read.spot` function and you can load this object as follow:

```
> data(Simon)
> ls()

[1] "Simon"

You have created an object of the class Spot in the current
environment so you will be able to apply to that object any
function of genArise that receives a spot as argument.
```

2.1 Spatial plots

Previous to any kind of transformation on the data once loaded the input, you are able to visualize it using the function `imageLimma`. This function constructs an image plot in a green to red scale representing the log_2 intensity ratio for each spot on the array. See Figure 1.

The imageLimma function receives several arguments depending on the data that is going to be analyzed. imageLimma is completly based in the imageplot function from the limma package.[1]

```
> data(Simon)
> datos <- attr(Simon, "spotData")# Extract spot data
> M <- log(datos$Cy5, 2) - log(datos$Cy3, 2)
> imageLimma(z = M, row = 23, column = 24, meta.row = 2,
  meta.column = 2, low = NULL, high = NULL)
```

[1] *Gordon K. Smyth (2004) "Linear Models and Empirical Bayes Methods for Assessing Differential Expression in Microarray Experiments ", Statistical Applications in Genetics and Molecular Biology: Vol. 3: No. 1, Article 3.* `http://www.bepress.com/sagmb/vol3/iss1/art3`

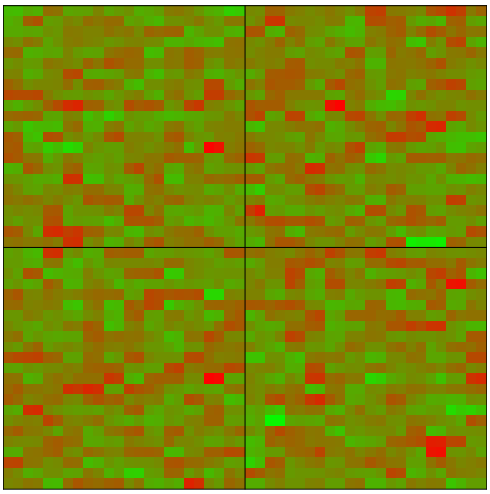

Figure 1: Image plot in a green to red scale.

```
Output: Plot the intensity values See Figure 1
```

A spatial plot of the background or Ratio-values can often reveal spatial trends or artifacts of various kinds. Figure 2 shows a spatial plot of red channel background for one array. Each spot on the array corresponds to one small square region on the plot. High background trends towards the edges of the plot stand out in the plot.

For example, the next code generates the plot for Cy3 Bg intensities shown in Figure 2.

```
> data(Simon)
> datos <- attr(Simon, "spotData")# Extract spot data
> R <- log(datos$BgCy5, 2)
> imageLimma(z = R, row = 23, column = 24, meta.row = 2,
  meta.column = 2, low = "white", high = "red")
```

```
Output: Cy5 intensity background plot. See Figure 2
```

And this is the code to generate the plot for green channel background shown in Figure 3..

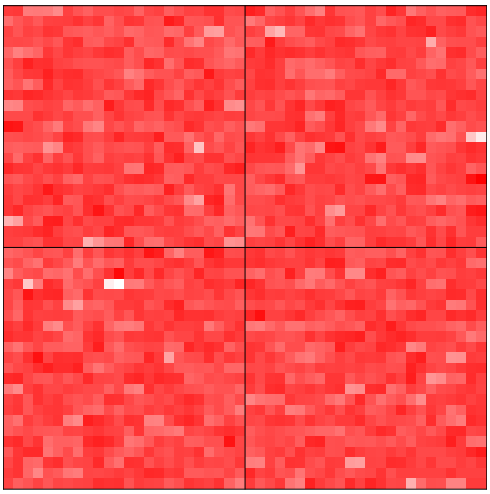

Figure 2: Cy5 intensity background preview.

```
> data(Simon)
> datos <- attr(Simon, "spotData")# Extract spot data
> G <- log(datos$BgCy3, 2)
> imageLimma(z = G, row = 23, column = 24, meta.row = 2,
  meta.column = 2, low = "white", high = "green")
```

```
Output: Cy3 intensity Background plot. See Figure 3
```

With the previous plots, you can see a preview of the raw data (without background correction and normalization). It is important to clarify that these image plot one does not replace the original TIFF image from the microarray experiment.

Data analysis requires to be able to plot a spot after apply any operation and genArise provides functions for this purpose. For example, you can plot the values R vs I, M vs A and Cy5 vs Cy3. This functions receive an object of the class Spot.

```
> data(Simon)
> ri.plot(Simon)
```

```
Output: R vs I plot. See Figure 4.
```

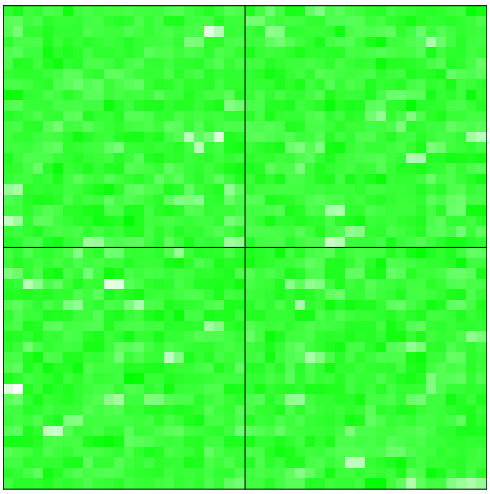

Figure 3: Cy3 intensity background preview.

```
> data(Simon)
> ma.plot(Simon)
```

```
Output: M vs A plot. See Figure 5.
```

```
> data(Simon)
> cys.plot(Simon)
```

```
Output: plot the log2(Cy3) and log2(Cy5) values. See Figure 6.
```

2.2 Transformation functions

At this moment, you can proceed with the data analysis. There are different functions for this purpose and all this functions will be described in this section.

2.2.1 Background Correction

The first of this functions makes the background correction; that is a subtraction Cy3 - BckgCy3 and Cy5 - BckgCy5 for each spot in the microarray. See Figure 7. The name of this function is `bg.correct`.

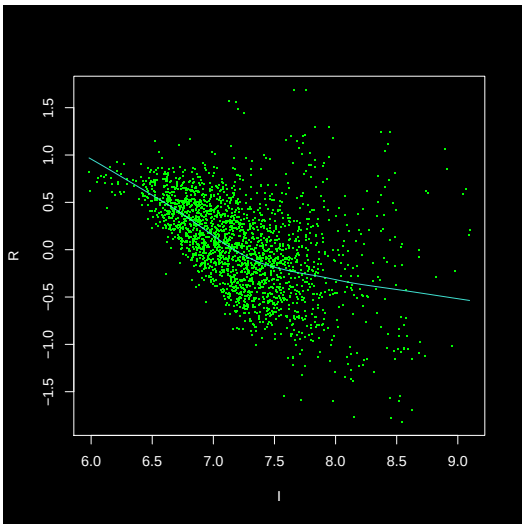

Figure 4: R vs I plot.

```
> data(Simon)
> c.spot <- bg.correct(Simon)
> ri.plot(c.spot)
```

```
Input: The argument of this function is a spot object
before background correction.
Output: A spot object after background correction. See Figure 7
```

2.2.2 Data Normalization

Data normalization can be done by grids or in a global way over all the microarray data, and each method may returns different results for the same spot object. We must remark that the grid normalization can just be applied to the complete data set, so you must not eliminate spots in order to avoid errors executing this function. In the normalization procedure any observation in which the R value is zero will be eliminated.

Grid normalization is executed with the function `grid.norm` and is mandatory to indicate the dimensions of the subgrid, so, you must indicate the number of rows and columns within each subgrid. See Figure 8.

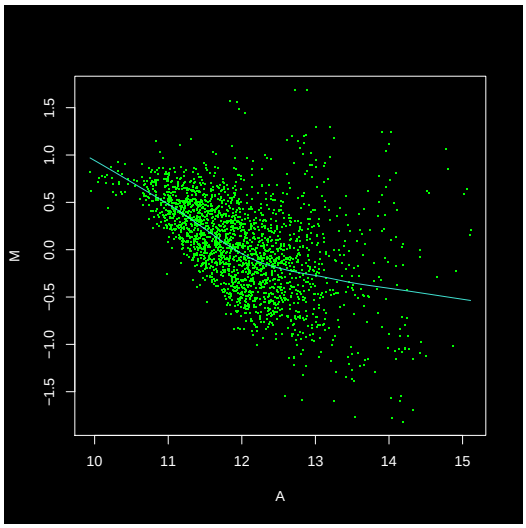

Figure 5: M vs A plot.

```
> data(Simon)
> n.spot <- grid.norm(mySpot = Simon, nr = 23, nc = 24)
> ri.plot(n.spot)

Input: The argument of this function is a spot object
       and the number of rows (nr) and columns (nc)
       of each subgrid.
Output: A normalized spot object. See Figure 8
```

On the other hand, the global normalization is executed with the function `global.norm` and it only requires as argument an object of the class Spot. See Figure 9.

```
> data(Simon)
> n.spot <- global.norm(mySpot = Simon)
> ri.plot(n.spot)

Input: The argument of this function is the spot object.
Output: A normalized (global normalization) spot object. See Figure 9
```

As we said previously both functions returns different results and for this reason you must choose the suitable function of normalization to the analysis

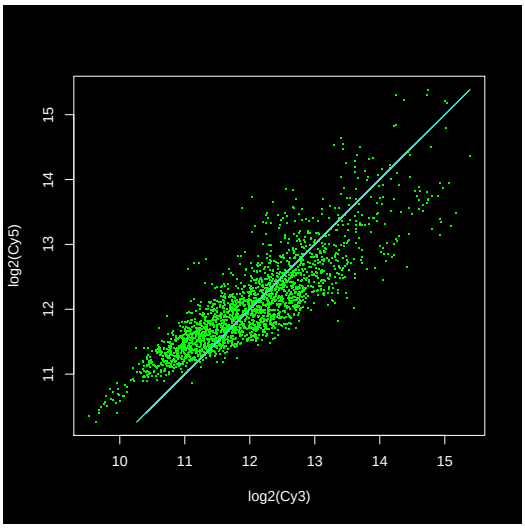

Figure 6: log2(Cy3) vs log2(Cy5) plot.

that you want to do. It for any reason spots are eliminated before normalization procedure (applying operations as filtering or duplicates elimination) the global normalization is the only normalization way.

In this example, the number of data in the spot is small and for this reason the obtained values with both normalization functions could seems very similar in the plots, however this is not the same in all the cases.

2.2.3 Data Filtering

Data filtering is an important step in the data analysis. genArise provide the intensity-based filtering algorithm described by John Quackenbush[2]. The name of the function is `filter.spot`. After this filtering you keep only array elements with intensities that are statistically significantly different from background. See Figure 10.

```
> data(Simon)
> f.spot <- filter.spot(mySpot = Simon)
```

[2] John Quackenbush "Microarray data normalization and transformation". Nature Genetics. Vol.32 supplement pp 496-501 (2002)

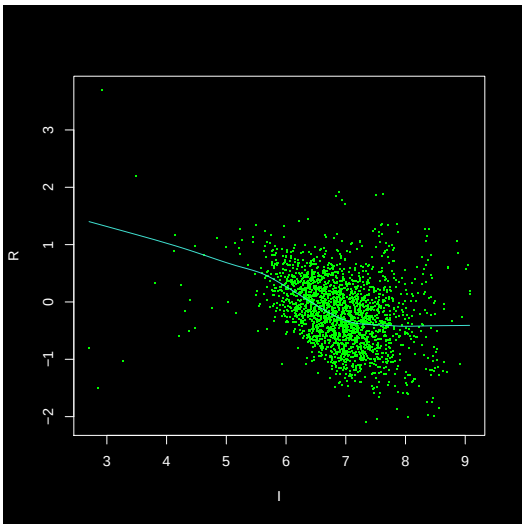

Figure 7: R vs I plot of background corrected data.

```
> ri.plot(f.spot)
```

```
Input: The argument of this function is a spot object.
Output: A filtered spot object. See Figure 10
```

2.2.4 Replicates Filtering

Another step is the replicates filtering, and in this step, a lot of observations are eliminated. One of the function for this purpose is called `spotUnique` and this function compute the geometric mean for the replicates to average our measurements, and the adjusted average measures for each gene can then be used to carry out further analyses. See Figure 11.

```
> data(Simon)
> u.spot <- spotUnique(mySpot = Simon)
> ri.plot(u.spot)
```

```
Input: The argument of this function is a spot object.
Output: A spot object without duplicates observations. See Figure 11
```

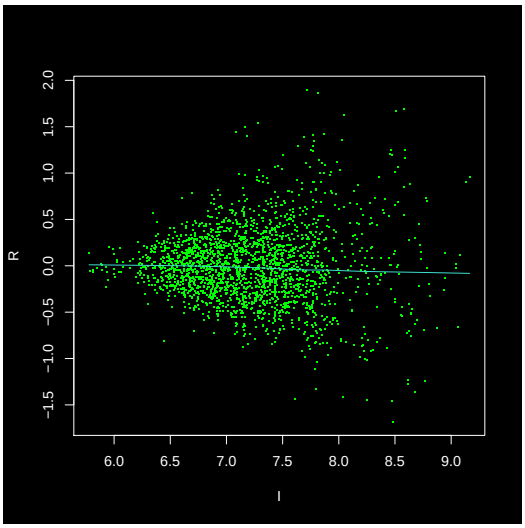

Figure 8: R vs I plot of grid normalized data.

However, genArise offers other functions for the same purpose, but with a different approach. We mean the function `alter.unique`. This function takes the R value of each one of the duplicated observations but just keep that observation with the extremest R value. So, if both of them are positives this function keeps the greater one, if both of them are negatives it keep the lower one and if there is one positive and one negative both observations are eliminated. It is clear that with this function a greater number of observations is conserved compared to those that are obtained with `spotUnique` function. By this way you will probably have at the final of the analysis a greater number of observations in the upper-expressed and lower-expressed because you keep the extreme values. See Figure 12.

```
> data(Simon)
> u.spot <- alter.unique(mySpot = Simon)
> ri.plot(u.spot)
```

```
Input: The argument of this function is a spot object.
Output: A spot object without duplicates observations. See Figure 12
```

A third function compute the mean for the duplicates. This function keeps not just one element for each id, but also this function eliminates those

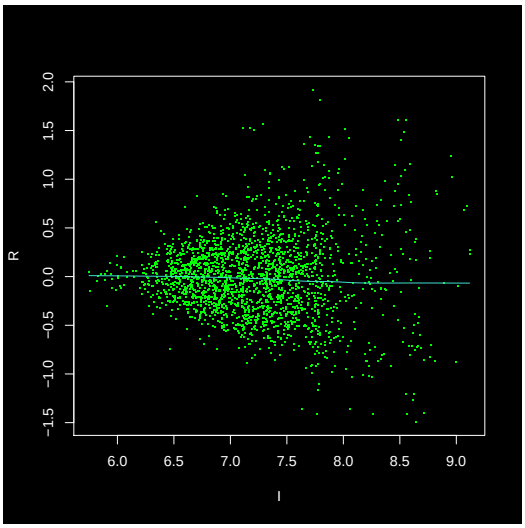

Figure 9: R vs I plot of data after global normalization.

points where the diference between the R value of the duplicated observations is bigger than 20

```
> data(Simon)
> u.spot <- meanUnique(mySpot = Simon)
> ri.plot(u.spot)
```

```
Input: The argument of this function is a spot object
Output: A spot object without duplicates observations. See Figure 13
```

2.3 Zscore

The approach we use involves calculating the mean and standard deviation of the distribution of log_2(ratio) values and defining a global fold-change difference and confidence; for this you must use a Z-score for the data set. In an R-I plot, that would be represented as parallel horizontal lines; genes outside of those lines would be the differentially expressed[2]. See Figure 14.

As a matter of fact genArise offers two options for this analysis. You just need to specify in the type argument on the `Zscore` function if you want a R-I or a M-A analysis. This function receive as argument an object of the

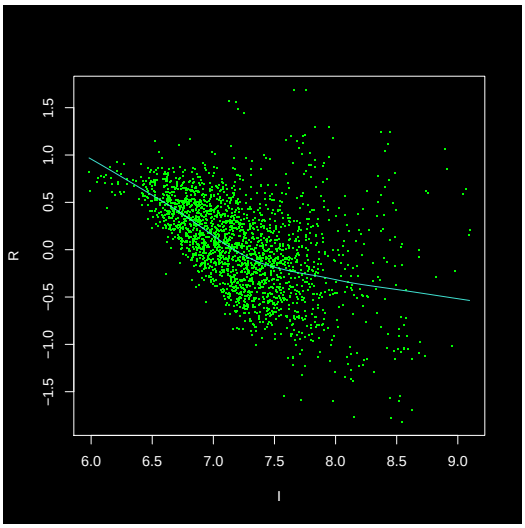

Figure 10: Filtered data, R vs I plot.

class Spot and returns an object of other class called *DataSet* that includes as one of their values the Z-score for the data set.

This is the example for `Zscore` using R-I values.

```
> data(Simon)
> s.spot <- Zscore(Simon, type="ri")

Input: The argument of this function is a spot object
Output: An object of the class DataSet
```

And this is the example for `Zscore` using `M-A` values

```
> data(Simon)
> s.spot <- Zscore(Simon, type="ma")

Input: The argument of this function is a spot object
Output: An object of the class DataSet
```

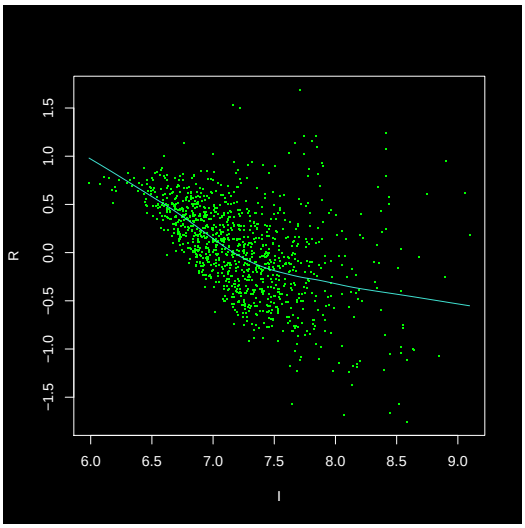

Figure 11: R vs I plot of data after apply `spotUnique` function.

2.4 Plotting DataSets

Since the objects of the `DataSet` class are different from the objects of the class Spot you need a function to plot them. For this purpose exists a function called `Zscore.plot`. So, in a R-I plot, array elements are color-coded depending on wether they are less than 1 standard deviation from the mean (green), between 1 and 1.5 standard deviations (blue), between 1.5 and 2 standard deviations (cyan), or more than 2 standard deviations from the mean (white). We use a subset of a data set called WT for the next examples. WT is a DataSet object that we have previously created.

```
> data(WT.dataset)
> Zscore.plot(WT.dataset)
```

```
Input: An object of the class DataSet
Output: Plot for identify differential expression. See Figure 14
```

2.5 genMerge

After you finish your slice analysis with function `Zscore` you get an up-regulated and down-regulated set. This will be the set of study genes for

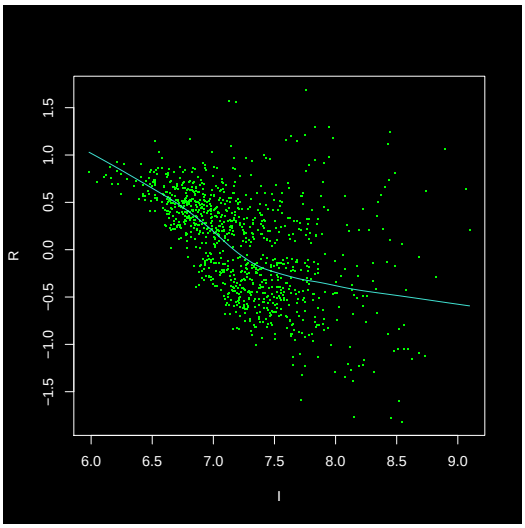

Figure 12: R vs I plot of data after apply `alter.unique` function.

genMerge. Given this set, the function `genMerge` retrieves functional genomic data for each gene and provides statistical rank scores for over-representation of particular functions in the dataset. The `genMerge` function is completly based on GeneMerge from Cristian I. Castillo-Davis and Daniel L. Hartl [3].

Given a set of genes (for example the upper-expressed), genMerge write in files the data of an statistic analysis of the upper-representation of functions or particular categories on the set of data upper-expressed.

You must take care in the format of the input file, the file must not have a header row. This function requires four data files and the file name where will be save the obtained information. The main file that the function needs are:

Association file. This file contains two columns splited by tab characters. The first column corresponds to the gene identifier in the microarray and the second one corresponds to a list of associated ids for the geneontol-

[3]Cristian I. Castillo-Davis, Department of Statistics, Harvard University `http://www.oeb.harvard.edu/hartl/lab/publications/GeneMerge`

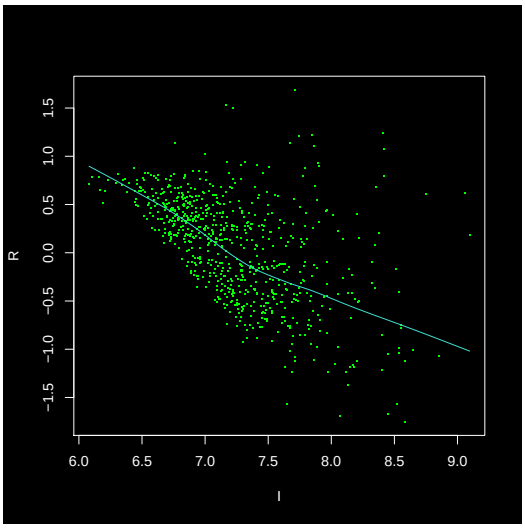

Figure 13: R vs I plot of data after apply `meanUnique` function.

ogy database to this gen (GO). Each GO in this list is separted by ";" and is necessary to consult a database to locate the associated GOs to each gen (see http://www.geneontology.org)

Example:

```
YAL001C GO:0003709;
YAL002W GO:0005554;
YAL003W GO:0003746;
YAL005C GO:0003754;GO:0003773;GO:0004002;
YAL007C GO:0005554;
```

Description file. This file also contains two columns splited by tab characters, the first one corresponds to the name of some GOs and the second one corresponds to the description associated to each GO in the database.

Example:

```
GO:0000005      ribosomal chaperone activity
GO:0000006      high affinity zinc uptake transporter activity
GO:0000007      low-affinity zinc ion transporter activity
GO:0000008      thioredoxin
GO:0000009      alpha-1,6-mannosyltransferase activity
```

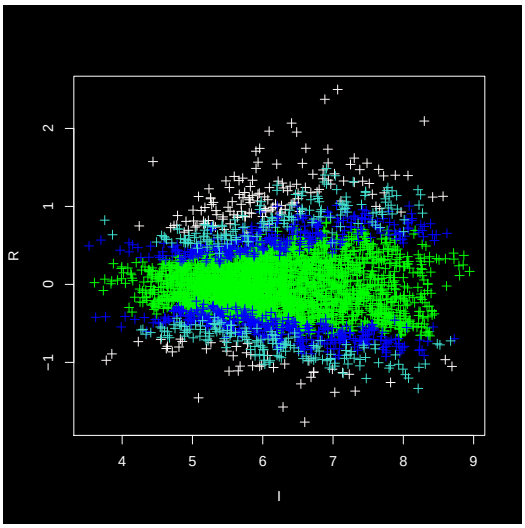

Figure 14: Data after Z-score Analysis.

```
GO:0000010      trans-hexaprenyltranstransferase activity
GO:0000014      single-stranded DNA specific endodeoxyribonuclease activity
GO:0000016      lactase activity
```

All genes (population.genes). This file should only contain one column, the spot identifiers or those of the input file not analyzed yet are the column information, each identifier is a row file.

```
YAL001C
YAL002W
YAL003W
YAL004W
YAL005C
YAL007C
YAL008W
YAL009W
YAL010C
YAL011W
```

The genes to study. Like the preceding described file, this file only contains one column and each row corresponds to the name of one of the genes in the set that will be studied (upper-expressed or lower-expressed).

2.6 Using genMerge

As the first step is create a spot object in the way described in the previous sections, the you must write the ids in a file, so you can do something like this:

```
> # Let's suppose that original spot is o.spot
> # To write the population file you can use write.table
> ids <- attr(o.spot, "spotData")$Ids
> ids <- unique(ids)
> write.table(ids, "population.genes")
```

`Output: File named population.genes that contains just the ids`

In the same way, with the `write.table` function you can write the identifiers of the genes set that will be studied. Suppose there exist a slice Spot called s.spot wich have the results after slice analysis, if you wish that your set of study have the upper-expressed identifiers and the lower-expressed identifiers to write the corresponding file, you must follow the next code.

```
> # To write the study genes file
> study.ids <- c(s.spot$Id.up, s.spot$Id.down)
> study.ids <- unique(study.ids)
> write.table(study.ids, "study.genes")
```

If you only want to preserve the identifier of some set, say upper-expressed or lower-expressed type the next lines:

```
> # Only write upper-expressed
> study.ids <- s.spot$Id.up # to write lower-expressed use s.spot$Id.down
> study.ids <- unique(study.ids)
> write.table(study.ids, "study.genes")
```

The other files should be constructed consulting a database or could be downloaded from the internet, this files should contain the format described above, so that the genMerge function operates correctly. The sintax of `genMerge` function is the next:

```
> genMerge(gene.association, description, population.genes,
  study.genes, output.file = "GeneMerge.txt"){
```

The results will be in the corresponding file, in the above example `GeneMerge.txt`.

2.7 Post-analysis

This function allows you to perform a set combinatorial analysis between the results previously obtained in different projects. This function is called `post.analysis` and it is mandatory that you have done the `Zscore` operation in all the selected projects. It is important to clarify that this function receives a list of files with extension `prj` as argument and for this reason you can't use it if the results to compare was not obtained by the genArise GUI.

The function `post.analysis` receives as argument a list of projects and look for the `zscore.txt` files for each project in the list. Once located each `zscore.txt` file the function get a list of ids for each project under the next criterion: all the ids must have a zscore value inside a range defined by the user.

Once obtained the ids list for each project a number of files with extension `set` are created in a directory. The name of this files consists in a sequence of 0 and 1. The number of digits in the file names is the same to the number of projects in the list passed as argument to the function. There is then, a relation between the number of digits in the file names and the projects. This relation is defined by the position specified in the file `order.txt` in the same directory you have passed as another argument in the function.

Each file will contain the ids that are present in all the sets (projects) in agreement to the file `order.txt` have 1 in the digit corresponding in the file name and in addition, this ids are not present in all the other sets (identified with a 0 in the filename).

3 The genArise GUI

To reduce the size of the genArise package in the text bellow the screenshots were not included. You can find the complete manual in the site of genArise `http://www.ifc.unam.mx/genarise`

For making the analysis easier, genArise joins all functions in a graphic interface, to call this function you must type the next line from the R prompt:

```
> genArise()
```

Now a menu-bar with two options and some icons for specific functions is displayed. When the mouse is passed over any icon in the main menu bar, a label with a description of the corresponding function will be displayed in the right side. From now on every icon in this menu bar will be identified by this description.

The GUI use R TclTk and if you are using the RGui in Windows, you can quickly notices that the RGui main window frequently "gets in the way", as genArise windows like to hide behind it. It therefore becomes convenient to bypass RGui altogether and just run Rterm and call `genArise()` from there.

3.1 Create a new project

You should create a New Project following the sequence **File** → **Project** → **New Project** in the menu, or click the icon **Create a new Project**.

A new window containing two panels (Input and Output) will be displayed. In the first of them you must specify the properties of the file containing the data, for example: location, was the experiment performed in the IFC?, microarray dimensions in other case. In this way, when the experiment was performed in the IFC you must choose the option **IFC** and the microarray dimensions will be automatically obtained.

In the input field labeled **Cy3**, **Cy5**, **BgCy3**, **BgCy5** y **Id**, you must type the number of the column in the file corresponding Cy3 intensity, Cy5 intensity, Cy3 background, Cy5 background and the gene identifiers, respectively.

In the inferior panel, there is an input field labeled with **Project Name** where you must type the name of the project to create and a new file with prj extension, as well as a directory with the specified name will be created. The next two input fields you must specify the name of the subdirectories from this project where the results files will be placed **(Location for Results)** and the plots **(Location for Plots)**. You can change this names writing on

the input field a new file name (without the path) or click the **browse** button.

Then, the window containing a preliminary view of the main grid is displayed as an image plot in a green to red scale representing the log_2 intensity ratio for each spot on the array. It is important to clarify that these image plot does not replace the original TIFF image from the microarray experiment.

You can also see the green and red levels by separated just selecting the corresponding option in the lower right part of the window.

In the upper right side panel of the window you can see the attributes of the file being analyzed and the experiment dimensions.

You can save the plot in a pdf file by selecting the option **Save as PDF** in the menu **Options** or clicking in the **Save graphic as PDF** icon. With the option **Note** of the same menu or clicking in the **Notes about the experiment** icon, a text editor will be launched where you can make some important notes through the analysis and save these in a file called notes.txt.

At this moment you can start the analysis by clicking on the menu **Analyze** or clicking in the **Make analysis data** icon, that will display a dialog box. Choosing **Yes** the program makes the operations in the following sequential order: background correction, loess normalization, intensity-filter, replicates filtering. Once this analysis is over you see the next window.

This window shows the graphic R -vs- I of the data obtained after the complete analysis. In the right side of the same window there are five options corresponding to the results of the operations performed through the analysis, choosing some of them the plot of the results for this step of the analysis will be displayed in the window.

genArise allows you to choose between three different graphic types (R -vs- I, M -vs- A o Cy3 -vs- Cy5) and you can display each one of this graphics just choosing the right option from the **Graphics** menu in the menu bar.

There is a white component in the lower side of the window where you can find the information for each one of the operations done.

At the end, you find one file for each operation done in the **Location for Results** subdirectory. All this files have a default name and you can't modify in no way this file name because they are essencial to be able to open this project with the genArise GUI in the future.

If you decide not to follow the wizard only the original data (no one operation performed) is displayed in a different window. In this case, you can perform any operation selecting the wished option in the menu bar. Once an operation is made, a new reference with the label corresponding to the operation performed will be showed up in the right side of the window and the graphic of the results obtained with that operation will be displayed. Is important to note that the operations are performed on the selected spot. If you wish to make a sequential analysis you must take care of the selected option.

3.1.1 Zscore Window

Clicking the **Zscore** option menu you can perform the operation Zscore described in section 2 and after select this option a new window will be displayed. You can display four plots for different zscore values in a range defined by the label of the corresponding option.

3.2 Open Project

You can open and consult the data of a previously done analysis with the option **Open Project** from the genArise main menu. After click this option you must select the prj file with the same name of the wished project from the file selector window. After this, a new window is displayed where you can check all plots from the results of the analyasis in that project. This plots are obtained from the data in the result files.

3.3 Post-analysis Window

This window allows you to obtain all the essencial information for the function `post.analysis` previously described. The list of the projects are choosed in the left side of the window. You can select several files with extension `prj`

one by one just clicking the Add button.

You must type and select the other arguments for the function: the directory that will contain all the output files, the range for the zscore and if the analysis will be done with ”up” or ”down” regulated.