Description of the maDB package

Johannes Rainer*

May 16, 2007

Tyrolean Cancer Research Institute
Innrain 66, 6020 Innsbruck, Austria, http://www.tcri.at
and Institute for Genomics and Bioinformatics, Graz University of Technology,
8010 Graz, Austria, http://www.genome.tugraz.at

Contents

1 Introduction

The *maDB* provides functionality to store (preprocessed) data along with sample informations and annotations from microarray experiments into a database, and to retrieve data sets or subsets of microarray experiments from a database generated with *maDB*. The relational concept of the database represented in figure 1 allows to store data of both two-color microarrays and single channel microarrays (Affymetrix GeneChips) into the same database. As database backend a PostgreSQL database is used, that allows the concurrent simultaneous access to a central database generated by *maDB* and or the usage of the microarray

*johannes.rainer@tcri.at

database by other tools (like the web application madbWeb written in PHP that allows to query a database created by *maDB* (http://madb.i-med.ac.at/madbWeb)). Another feature of *maDB* allows to search for genes with a specific gene expression or regulation pattern over a set for samples (arrays). As expression or regulation pattern template a custom template or the expression/regulation pattern of a specific gene can be used (more informations in section 5).
In addition to the database functionality *maDB* provides various functions for microarray data visualizations like MA plots, volcano plots and others (see section 6 for some examples).
The database model of a maDB database is shown in figure 1. Attributes with the ending *_pk* and *_fk* represent primary keys and foreign keys respectively. Briefly each microarray experiment (*experiments* table) consists of a set of arrays (*arrays* table), where each of the arrays has one (Affymetrix) or more (two or more color arrays) signal channels (*signal_channels* table). On each signal channel of an array a sample is hybridized (connection of the *signal_channels* table to the *samples* table) and a set of n intensity values, where n is the number of features on the array (eg spots on a two-color microarray or the number of probesets of an Affymetrix GeneChip array), are measured for each signal channel. These intensity values (in fact already preprocessed expression values) of a signal channels are stored into the *exp_values* table.
The expression values of the features of two signal channels / samples can be compared by calculating M values (log2 fold change values, regulation values) and A values (average expression values) with the `dbCalculateRegulations` function. Such M and A values are stored in the *regulation_values* table. The *comparisons* table contains the information which signal channels / samples are compared in a comparison and establishes the linkage to the regulation values.
The annotation of features can be stored into the *annotation* table. Annotation should be inserted / updated / mantained using the `dbUpdateAnnotation`. Some annotation columns are required, other can be added dynamically (depending on the annotation table submitted to the `dbUpdateAnnotation` function).

2 Storing a microarray experiment into the database

To establish a connection to a running PostgreSQL backend database the `dbConnect` function from the *RdbiPgSQL* package has to be used, the function `dbSendQuery` can be used to send SQL commands to the database backend (the `dbGetResult` function allows to fetch results back to R). In the code below first the database that should be used by maDB is created (using the `CREATE DATABASE` SQL command), and then a connection to the newly generated database is created. Setting maDBs logging level *log.level* to "ERROR" avoids that maDB logs every SQL call into the log file (which can become huge).

```
> library(maDB)
> con.su <- dbConnect(PgSQL(), host = "localhost", user = "postgres",
```

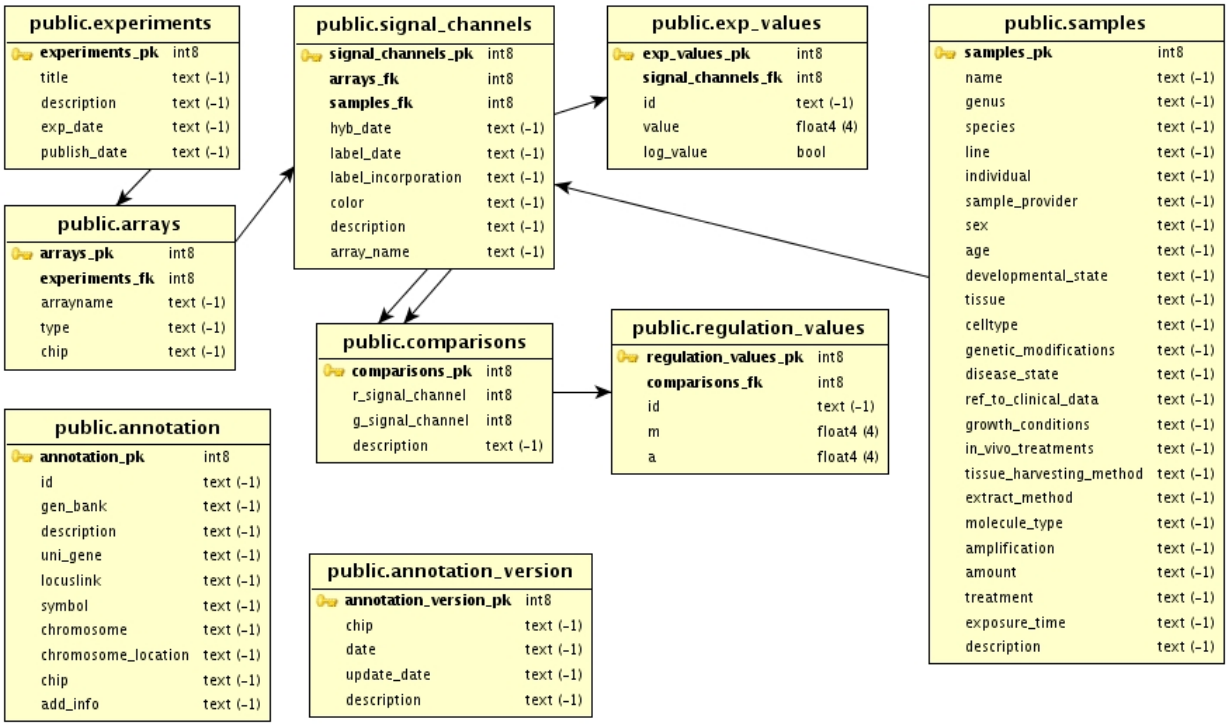

Figure 1: The maDB database model. A short description is given in the main text.

```
+     dbname = "template1")
> result <- dbGetResult(dbSendQuery(con.su, "SELECT datname FROM pg_database"))
> if (sum(result[, "datname"] == "madb") > 0) {
+     dbSendQuery(con.su, "DROP DATABASE madb")
+ }

status = 1
status.string = PGRES_COMMAND_OK
rows = 0
columns = 0
is.binary = FALSE
command.response = DROP DATABASE

> dbSendQuery(con.su, "CREATE DATABASE madb")

status = 1
status.string = PGRES_COMMAND_OK
rows = 0
columns = 0
is.binary = FALSE
command.response = CREATE DATABASE

> dbDisconnect(con.su)
> con <- dbConnect(PgSQL(), host = "localhost", user = "postgres",
+     dbname = "madb")
> log.level <- "ERROR"
```

In *real life* connections should not be established as user *postgres*, and the PostgreSQL database should be configured to require password veryfication.

All database tables are automatically created by the `publishToDB` function upon insertion of a first data set. As an example the *smallALL* dataset of the *maDB* package is used. This dataset contains already preprocessed (gcrma) data from 8 Affymetrix CEL files (samples from 4 children with ALL (acute lymphoblastic leukemia, 2 children with B-ALL, 2 with T-ALL), two peripheral blood samples per children, one sample before and one after 6 hours in vivo treatment with glucocorticoids (GC), dataset is reduced to 1000 probesets). The preprocessing was performed using the `gcrma` function which returns an `ExpressionSet` object (like all functions from the *affy* package). This object is first casted into a `MadbSet` object, which directly extends the `ExpressionSet` object from the *Biobase* package. Also objects from the *limma* package (`RGList`, `MAList`) for two color microarrays can be transformed into a `MadbSet` by using the `newMadbSet` function.
`MadbSet` objects inherit all functionality from the `ExpressionSet` and provide some additional methods like the database storage/retrieval functions or some plotting functions.

```
> data("smallALL")
> class(NormChips)

[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"

> NormChips <- newMadbSet(NormChips)
> class(NormChips)

[1] "MadbSet"
attr(,"package")
[1] "maDB"
```

Before storing the experiment into the database it is useful to describe the samples that were hybridized onto the chips. This information can be stored into a list of `Samples` objects that can be added to the `@samples` slot of the `MadbSet` object. The linkage between signal channel and samples (i.e. which sample was hybridized on which signal channel/ array) has to be established with `SignalChannel` objects.
The `publishToDB` method finally stores the data set into the database.

```
> Sample1 <- new("Sample", name = "B-ALL-13", individual = "13KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample2 <- new("Sample", name = "B-ALL-13", individual = "13KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample3 <- new("Sample", name = "B-ALL-17", individual = "17KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample4 <- new("Sample", name = "B-ALL-17", individual = "13KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample5 <- new("Sample", name = "T-ALL-20", individual = "20KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample6 <- new("Sample", name = "T-ALL-20", individual = "20KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
+     exposure.time = "6h")
> Sample7 <- new("Sample", name = "T-ALL-25", individual = "25KKI",
+     tissue = "PB", species = "hs", exposure.time = "0h")
> Sample8 <- new("Sample", name = "T-ALL-25", individual = "25KKI",
+     in.vivo.treatments = "GC", tissue = "PB", species = "hs",
```

```
+     exposure.time = "6h")
> TheSamples <- list(Sample1, Sample2, Sample3, Sample4, Sample5,
+     Sample6, Sample7, Sample8)
> SigChannels <- getSignalChannels(NormChips)
> for (i in 1:length(SigChannels)) {
+     SigChannels[[i]]@sample.index <- i
+ }
> publishToDB(NormChips, con, exp.name = "maDB_Vignette", signal.channels = SigChannels,
+     samples = TheSamples, preprocessing = "gcRMA", v = FALSE)
```

Regulation values (M values) between signal channels (samples) in the database can be calculated using the `dbCalculateRegulations` functions, that takes the name of the dataset/experiment in the database and the indizes of the signal channels that should be compared as input parameters, calculates the M and A values and inserts them into the database. These regulation values are then available for further analyses, or for later export from the database.
The function `dbGetExperimentInfo` can be used to get a list of available experiments that are stored in the database, or to retrieve further informations like the arrays, signal channels and samples from a specific experiment. This information can then be used to decide which experiments/subset of a experiment should be fetched from the database, or, just like in the code below, for which comparisons between samples regulation (M and A) values should be generated.

```
> dbGetExperimentInfo(con)

You have not submitted an experiment title, please submit one of the following that are available in the database:
  experiments_pk         title description exp_date preprocessing pubmed_id
1              1 maDB_Vignette             18011977         gcRMA
              publish_date
1 Wed May 16 16:46:57 2007

> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> colnames(info)

 [1] "arrays_pk"          "chip"               "arrays_fk"
 [4] "signal_channels_pk" "array_name"         "samples_fk"
 [7] "name"               "individual"         "line"
[10] "tissue"             "treatment"          "in_vivo_treatments"
[13] "exposure_time"

> info[, c("name", "in_vivo_treatments", "exposure_time")]

      name in_vivo_treatments exposure_time
1 B-ALL-13               <NA>            0h
2 B-ALL-13                 GC            6h
3 B-ALL-17               <NA>            0h
4 B-ALL-17                 GC            6h
5 T-ALL-20               <NA>            0h
6 T-ALL-20                 GC            6h
7 T-ALL-25               <NA>            0h
8 T-ALL-25                 GC            6h
```

Below regulation values are calculated by comparing the expression values from the 1000 probesets of the chips after 6 hours GC-treatment with those before treatment. These regulation values are stored into the `regulation_values` database table.

```
> dbCalculateRegulations(con, "maDB_Vignette", comparisons = list(c(2,
+     1), c(4, 3), c(6, 5), c(8, 7)), v = FALSE)
```

To get an information of the comparisons that are available for a microarray experiment, the `dbGetComparisons` can be used.

```
> Comparisons <- dbGetComparisons(con, "maDB_Vignette")
> Comparisons[, c("name.red", "exposure_time.red", "name.green",
+     "exposure_time.green")]

     name.red   exposure_time.red name.green exposure_time.green
[1,] "B-ALL-13" "6h"              "B-ALL-13" "0h"
[2,] "B-ALL-17" "6h"              "B-ALL-17" "0h"
[3,] "T-ALL-20" "6h"              "T-ALL-20" "0h"
[4,] "T-ALL-25" "6h"              "T-ALL-25" "0h"
```

3 Reloading a set of arrays from the database

The function `loadFromDB` can be used to load a dataset (microarray experiment) or data from a subset of arrays from the database. In the example below only the arrays with samples from patient T-ALL-25 are fetched from the database (with *loadFromDB(newMadbSet(),con,"maDB_Vignette")* the whole dataset would be loaded).

```
> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> info.reduced <- info[info[, "name"] == "T-ALL-25", ]
> TALL25 <- loadFromDB(newMadbSet(), con, "maDB_Vignette", pk = info.reduced[,
+     "signal_channels_pk"], v = FALSE)
> TALL25

MadbSet (storageMode: lockedEnvironment)
assayData: 1000 features, 2 samples
  element names: exprs
phenoData
  rowNames: T-ALL-25-0h.CEL, T-ALL-25-6h.CEL
  varLabels and varMetadata:
    sample: sample numbering
featureData
  featureNames:
  varLabels and varMetadata: none
experimentData: use 'experimentData(object)'
Annotation [1] "hgu133plus2"
```

3.1 Loading expression values for a set of genes from the database

The function `getEDB` can be used to fetch the expression values from a set of features (in a set of samples) from the database. The code below loads the expression values for the first 5 probe sets in the samples of patient B-ALL-13 from the database. To load the data from the correct signal channels/arrays, the primary keys of the appropriate signal channels have to be submitted.

```
> info <- dbGetExperimentInfo(con, "maDB_Vignette")
> info.reduced <- info[info[, "name"] == "B-ALL-13", ]
> EValues <- getEDB(con, id = rownames(exprs(NormChips))[1:5],
+     signal.channels.pk = info.reduced[, "signal_channels_pk"],
+     v = FALSE, column.names = c("name", "exposure_time"))
> EValues
```

```
          B-ALL-13, 0h B-ALL-13, 6h
1007_s_at     6.980063     6.609214
1053_at       6.340858     5.607547
117_at        2.811279     2.997891
121_at        3.163045     3.177225
1255_g_at     2.185753     2.200411
```

3.2 Loading regulation values for a set of genes from the database

The function to load regulation values for a set of genes from a set of comparisons from the database is called `getMDB` and works just like the `getEDB` function (with the exception that instead of the promary keys of the signal channels, those of the comparisons have to be submitted).

```
> info <- dbGetComparisons(con)
> info.reduced <- info[info[, "name.red"] == "B-ALL-13" | info[,
+     "name.red"] == "B-ALL-17", ]
> MValues <- getMDB(con, id = rownames(exprs(NormChips))[1:5],
+     comparisons.pk = info.reduced[, "comparisons_pk"], v = FALSE,
+     column.names = c("name", "exposure_time"))
> MValues

          UniqueID    B-ALL-13, 6h B-ALL-17, 6h
1007_s_at "1007_s_at" "-0.370849"  "0.130771"
1053_at   "1053_at"   "-0.733311"  "0.192474"
117_at    "117_at"    "0.186612"   "-0.031165"
121_at    "121_at"    "0.01418"    "-0.041603"
1255_g_at "1255_g_at" "0.014658"   "-0.003137"
```

4 Inserting the annotation into the database

To insert the annotation for the features the function `dbUpdateAnnotation` can be used. This function can also be used to update the annotation in the database (the old annotation will be backed up automatically).
The annotation created in this example bases on the annotation of the *hgu133plus2* meta data package and the annotation table is created with the functions of the *annaffy* package.

```
> IDs <- dbGetResult(dbSendQuery(con, "SELECT DISTINCT id FROM exp_values"))[,
+     "id"]
```

The code above gets a unique list of IDs from the `exp_values` database table.

```
> library(annaffy)
> AnnotationTable <- matrix(ncol = 4, nrow = length(IDs))
> colnames(AnnotationTable) <- c("id", "gen_bank", "uni_gene",
+     "symbol")
> AnnotationTable[, 1] <- IDs
> AnnotationTable[, 2] <- getText(aafGenBank(IDs, "hgu133plus2"))
> AnnotationTable[, 3] <- getText(aafUniGene(IDs, "hgu133plus2"))
> AnnotationTable[, 4] <- getText(aafSymbol(IDs, "hgu133plus2"))
> dbUpdateAnnotation(con, data = AnnotationTable, chip = NormChips@annotation,
+     v = FALSE)
```

This (short) annotation table can be inserted using the `dbUpdateAnnotation` function.

The annotation of feature IDs using annotation information from the maDB database can be performed with the `dbGetAnnotation` or `getAnnotation` functions.

```
> dbGetAnnotation(con, id = IDs[1:5], v = FALSE, chip = "hgu133plus2")

     id          gen_bank description uni_gene               locuslink symbol
[1,] "1007_s_at" "U48705" NA          "Hs.593011, Hs.631988" NA        "DDR1"
[2,] "1053_at"   "M87338" NA          "Hs.647062"            NA        "RFC2"
[3,] "117_at"    "X51757" NA          "Hs.352642"            NA        "HSPA6"
[4,] "121_at"    "X69699" NA          "Hs.469728"            NA        "PAX8"
[5,] "1255_g_at" "L36861" NA          "Hs.92858"             NA        "GUCA1A"
     chromosome
[1,] NA
[2,] NA
[3,] NA
[4,] NA
[5,] NA
```

Another way to query the annotation table in the database is shown below, where all probesets targetting the same gene are retrieved.

```
> dbGetAnnotation(con, id = "CD24", chip = "hgu133plus2", search.col = "symbol",
+     v = FALSE)

  id            gen_bank description uni_gene               locuslink symbol
1 "208651_x_at" "M58664" NA          "Hs.644105, Hs.652291" NA        "CD24"
2 "266_s_at"    "L33930" NA          "Hs.644105, Hs.652291" NA        "CD24"
  chromosome
1 NA
2 NA
```

5 Searching for genes with similar expression or regulation pattern

The function `dbSearchSimilarPattern` compares expression or regulation pattern of a template gene (or a artificial pattern) with those of all other genes in the database. The similarity is measured (like in the HCL (hierarchical clustering) analysis) using different distance measurement methods (euclidian, pearson correlation ...).

In the example below genes with a similar expression pattern like the CD19 gene (which is a B-lymphocyte antigen) in the 8 samples stored in the database are identified. The parameter `include.by.name` allows to specify the samples, that should be included in the similarity calculation (in our case all 4 patients (and therefore all 8 samples)). The default similarity measure is the euclidian distance, but also other similarity measurement methods can be used.

As result a `Similarity` object is returned. This object contains all calculated similarity measures.

```
> CD19Sim <- dbSearchSimilarPattern(con, id = "206398_s_at", include.by.name = c("T-ALL-20",
+     "T-ALL-25", "B-ALL-13", "B-ALL-17"), include.values = TRUE,
+     v = FALSE)
> dbGetAnnotation(con, id = names(CD19Sim@distances)[1:5], chip = "hgu133plus2",
+     columns = "symbol", v = FALSE)
```

```
     id             symbol
[1,] "206398_s_at"  "CD19"
[2,] "208651_x_at"  "CD24"
[3,] "1553102_a_at" "CCDC69"
[4,] "1552274_at"   "PXK"
[5,] "266_s_at"     "CD24"

> dbDisconnect(con)
> con <- NULL
```

6 Microarray data visualizations

The function `drawMA` can be used to draw MA plots from objects of the type `MadbSet`. The parameters r and g represent the index of the signal channel in the `exprs` slot that should be used as *red* and *green* signal channel. The figure 3 represents the MA plot of the 100 probe sets of the 6 hours GC treated sample of patient T-ALL-25 against the sample before treatment.

```
> plot(CD19Sim)
```

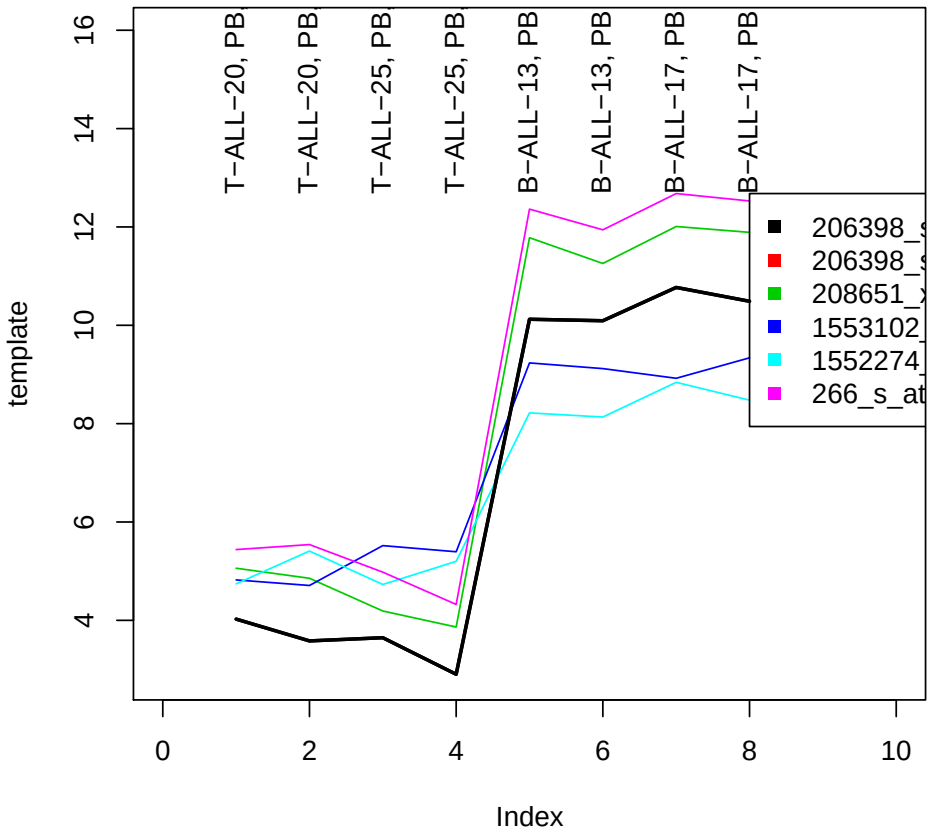

Figure 2: Genes with similar expression pattern as CD19. (black line: expression pattern of the probe set 206398_s_at (CD19), the other lines correspond to the 5 genes with the most similar expression patterns).

```
> drawMA(TALL25, r = 2, g = 1, colramp = topo.colors)
```

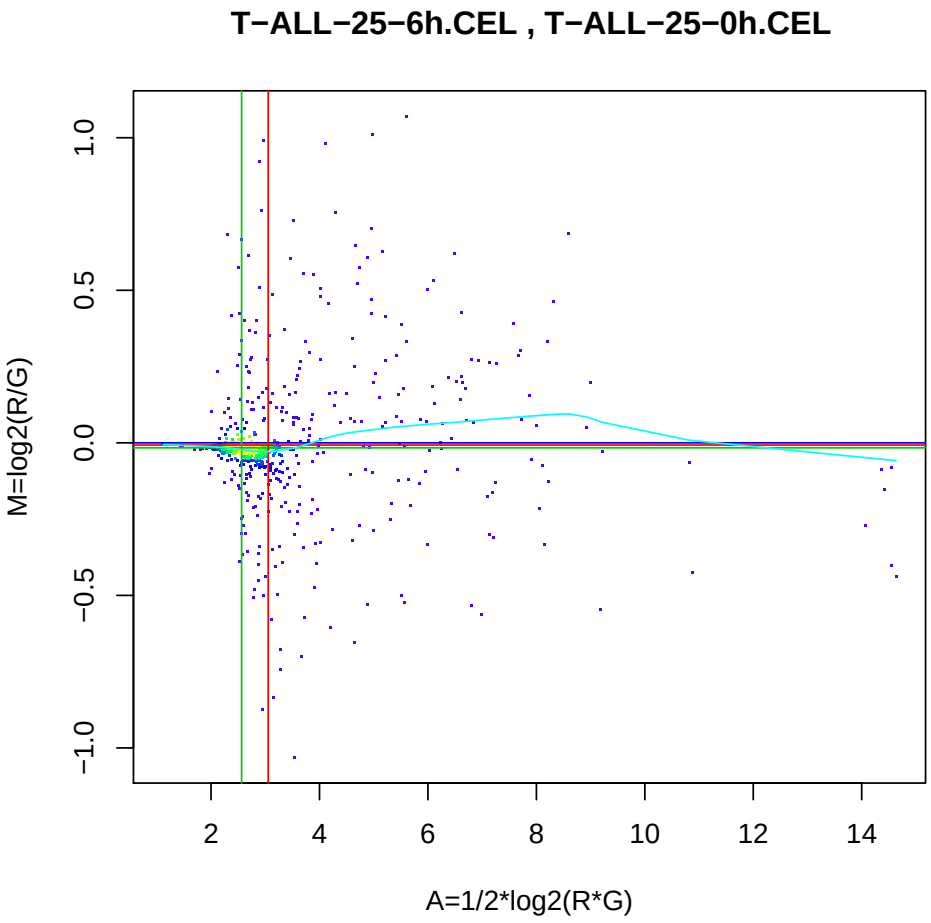

Figure 3: MA plot of patient T-ALL-25 samples. Data points are colored according to the local point density