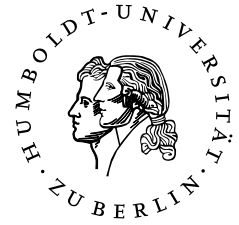


HUMBOLDT-UNIVERSITÄT ZU BERLIN



Statistical classification and programming for Microarrays

Adrien du Moulinet d'Hardemare
404085

PD. Marlene Müller
Professor Bernd Rönz

2005

Lehrstuhl für Statistik

Thanks:

I would like to thank M. Müller, P. Ecochard and E. Haquet for their precious help

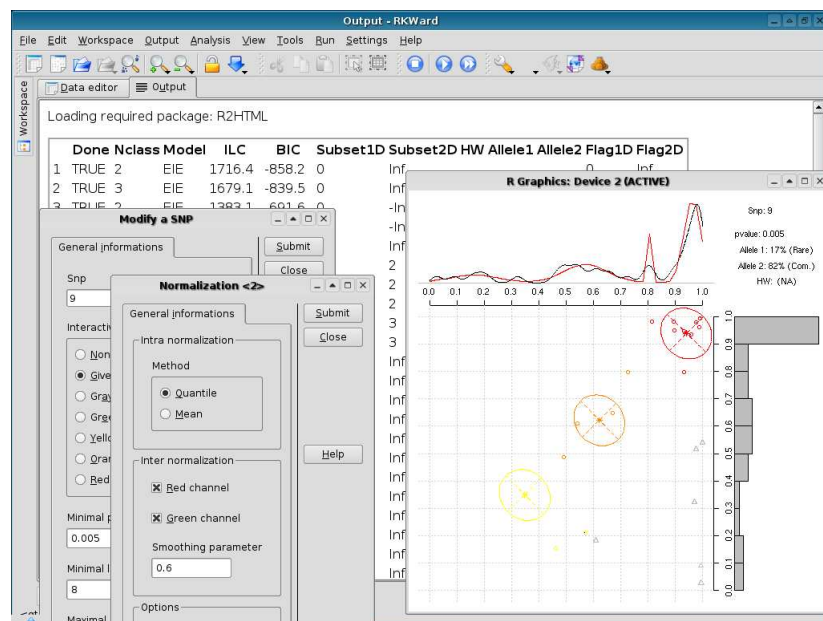


Figure 1: Snapshot of RKsnp

Contents

Introduction	1
1 Biological background	3
1.1 Down Syndrome disease	3
1.2 Single Nucleotid Polymorphisms	5
1.3 The microarray technology	5
1.4 Presentation of the data	7
2 Normalization of the data	11
2.1 LOESS normalization	12
2.2 Mean normalization	14
2.3 Quantile normalization	14
2.3.1 First algorithm	15
2.3.2 Second algorithm	15
3 Classification of the data	19
3.1 Modeling the data	19
3.2 The EM algorithm	20
3.2.1 Implementation of the algorithm	22
3.2.2 Various models	23
3.3 Complements to the EM algorithm	25
3.3.1 Choosing the best model	25
3.3.2 Subsetting data	27
4 Clustering software	29
4.1 Statistical computation	30
4.1.1 Programming with R	30
4.1.2 The 'snp' package	31
4.2 The Graphical User Interface	33
4.2.1 What is RKward	33
4.2.2 Architecture of a RKplugin	34

4.2.3	Presentation of 'RKsnp'	38
	Conclusion	43
A	User Guide for the R Package	49
A.1	Installation of the package	49
A.2	Handling input	50
A.3	Normalization of the data	52
A.3.1	Problem	52
A.3.2	Within and between normalizations	52
A.4	Classify SNP	54
A.4.1	Select a type	54
A.4.2	The makeSnp function	55
A.4.3	Access classification	57
A.5	Options	59
A.5.1	Update	59
A.5.2	General options	61
B	Help files	63

Introduction

This master thesis presents the statistical work involved in the biological process known as "microarray technology". The microarray technology is a recent tool used in genetics and involves a lot of statistical work to control the reliability of assessment so as to use it safely for medical purpose.

At first, microarrays were applied to cDNA process (transcriptomics) in which statistical problems are well known: normalization of the data, stabilization of the variance and assessing differentially expressed genes. However, our study focuses on a more recent application of the microarray technology, i.e. applied to **Single Nucleotid Polymorphisms** (SNPs). SNPs analysis have promoted a recent breakthrough in genetic research. Whereas transcriptomics analyze how many proteines a gene produces at a specific moment by quantifying the amount of cDNA in a cell, SNPs analysis deals with local mutations in a DNA sequence. On one hand, we study the "expression" of a gene, on the other hand, we are interested in genotyping people. The major advantage of such a technology is based on a simple idea: thanks to SNPs, one can characterize a genome and therefore try to highlight a correlation between genomes with similar features and the phenotype¹ of people. For example, in the present work, we are trying to classify trisomic and disomic people according to 54 SNPs. If a class of genomes with similar mutations can be crossed with a disease observed among trisomic people, one may assume that at least one of the 54 SNPs is a source of genetic disturbances involved in the Down Syndrome disease. The corresponding gene can be isolated and studied further.

Actually, the main statistical problem in such an analysis can be summed up in this question: how can we classify ones according to their SNPs so as to highlight a correlation between genomic and phenotypic features? This master thesis deals mainly with the genotyping problem. Two types of people will be studied: disomic and trisomic people. The main group of interest is trisomic people, but to tackle the problem of the quality of the measures and

¹The phenotype of a people is defined by all his observed features (IQ, color of hair, blood properties...)

to answer other genetic questions, disomic people will also be analyzed..

Apart from the statistical reliability of the genotyping, another problem of this study is the reproducibility of analysis. Our data are only a training set and other data should arrive next year. Therefore, biologists, who usually know little programming, will have to redo this analysis. This leads to the implementation of a documented software. This software is endowed with the R package facilities (help and example files) and with a Graphical User Interface working under Unix.

In the first chapter, some concepts concerning Down Syndrome Disease and SNPs are briefly explained to give a clue of what is at stake. Since the complexity of the microarray technology induces bias in observations that will have to be removed, the biological experiment is also presented.

The second chapter deals with the problem of normalization of measures. As hinted, observations issued from a microarray technology experiment cannot be at first analyzed and must be "filtered" from background noise. A two step analysis, "within" and "between" normalization, is implemented to this purpose. Different methods are described.

The third chapter describes the analysis that will give the value of each SNP to genotype each people. A gaussian mixture model is implemented and thanks to an EM algorithm, data are classified. Since SNPs have different qualities and different structures, three various models using this algorithm are proposed and a criterion to choose the best model is also described.

In a fourth chapter, we describe how this analysis has been implemented in a software and how to use it. The software should be both robust and comfortable: therefore, the implementation has been divided in two parts according to those two main goals. The first part is the core program (statistical computations, experiment adapted functions) with the R language. The program works both under Unix and Windows. The second part is the implementation of the Graphical User Interface in C++, XML and bribes of php.

Chapter 1

Biological background

Using SNPs to analyze Down Syndrome disease is innovative in two ways. First, it aims at promoting a new conception of Down Syndrome which undertones therapeutic goals thanks to genetic research. Secondly, the SNPs analysis is here used on a very big scale (several thousands of people instead of only a few people as usual).

To understand what is at stake in the present study, the biomedical concepts used in the SNPs experiment are explained. First, some words are consecrated to the Down Syndrome disease and describe the hypothesis that validates such an experiment. Then the very concept of SNPs is detailed. Microarray technology's principles are also briefly explained to understand how observations can be classified so as to genotype people. At last, a paragraph is devoted to the data set used in the present study.

1.1 Down Syndrome disease

Down Syndrome (also called Down's Syndrome) encompasses a number of genetic disorders, of which trisomy 21 (a non-disjunction) is the most representative, causing highly variable degrees of learning difficulties and physical disabilities. It is named after John Langdon-Down, the British doctor who first described it. Down syndrome or trisomy 21 is the most common chromosomal abnormality (1 per 660 births). It is confirmed by the study of the karyotype, which points out three exemplars of the chromosome 21, instead of two.

Several types of trisomy 21 are to be distinguished. The "Free Trisomy" is defined by the fact that the 21th chromosome is three times entirely duplicated. This type of trisomy concerns 95 % of patients. It generally results from a non disjunction of the chromosome 21 during the meiosis. This er-

ror takes place during the first meiotic division. However, some cases are provoked by an error during the mitosis, when homologous chromosomes replicate, in a more or less advanced step during the embryo development, which has as main result the presence in the human being of normal and trisomic cells. A third type of trisomy, "Partial trisomy", less frequent, is used when only a part of the chromosome 21 is duplicated three times.

The Down Syndrome patients phenotype can be characterize by following features: Down syndrome children's IQ is rarely measured above 60. Down syndrome children's brains are usually small and underweight. The cerebellum and brain stem are unusually small, as is the superior temporal gyrus. Educational progress may also be damaged by illness and disabilities, such as recurring infectious diseases, heart problems, poor eyesight, and hearing problems. Other physical characteristics associated with disorder include presence of a simian crease. Early educational intervention, screening for common problems such as thyroid functioning, medical treatment, a conducive family environment, vocational training, etc. can improve the overall Down syndrome children's development (See http://en.wikipedia.org/wiki/Down_syndrome).

However, it must be noted that phenotypic features vary among Down Syndrome patients. The mental handicap can vary from severe to light and facial dysmorphisms can also vary among people. Those differences cannot be all explained by education or environmental factors and must therefore find their causes in a genetic abnormality varying among people.

The first hypothesis tackling this idea has been called "development instability". Phenotypic disturbances are said to be caused by the over-numerous chromosome 21. The idea is basically as follows: genes are used to produce proteins. Since trisomic cells do have three chromosomes 21 instead of only two, trisomic cells are supposed to produced an irregular amount of the needed proteins. Hence, Trisomy 21 results in over-expression of genes located on chromosome 21.

However, several arguments can be opposed to this hypothesis. First this "development instability thesis" cannot take into account the great differences observed in trisomic phenotypes. Moreover, phenotypic disturbances are not correlated with the size of the duplicated part of the chromosome 21 but with the genes included in the duplicated part.

Therefore, another hypothesis, called "genetic dose" says that only a small amount of genes in the chromosome 21 causes phenotypic disturbances when duplicated three times. For example, the abnormality of the "interventricular septum" has been observed to be often associated with an heterotrissomy of a chromosomal region containing the DS-CAM gene (See [Baptista]).

1.2 Single Nucleotid Polymorphisms

To highlight differences between genomes of trisomic people and to characterize their set of genes, Single Nucleotid Polymorphisms are analyzed in this experiment. Single nucleotide Polymorphisms or SNPs (pronounce "snips") are DNA sequence variations that occur when a single nucleotide (A,T,C,or G) is altered in the genome sequence. For example a SNP might change the DNA sequence AAGGCTAA to ATGGCTAA (note that a SNP can take only two values, here A or T). To be considered as a SNP, a variation must occur in at least 1% of the population. SNPs, which make up about 90% of all human genetic variation, occur every 100 to 300 bases along the 3-billion-base human genome. Two of every three SNPs involve the replacement of cytosine (C) with thymine (T). SNPs can occur in both coding (gene) and non-coding regions of the genome.

Many SNPs have no effect on cell functions, but scientists believe some could predispose people to disease or influence their response to a drug. Although more than 99% of human DNA sequences are the same across the population, variations in DNA sequence can have a major impact on how humans respond to disease. This makes SNPs of great value for biomedical research and for developing pharmaceutical products or medical diagnostics. SNPs are also evolutionarily stable –not changing much from generation to generation –making them easier to follow in population studies. Therefore, in the present analysis trisomic and disomic people are studied, the disomic people being parents of the trisomic patients.

The first difficulty with SNP occurs when one wants to determine a particular SNP, i.e. to find out which basis (A, C, G, or T) a chromosome has wherein the mutation is supposed to take place. Recalling that disomic (trisomic) people have two (three) chromosomes, hence, two (three) sequences of DNA, a SNP for a disomic (trisomic) people can be characterized by three (four) classes (See graphic 1.1 page 6). Let's say, for example, we want to determine a SNP, eg a DNA sequence where the usually observed Thymine can be replaced by a Cytosine. Disomic people can be divided into three classes: those who have two thymine (class called TT), those who have two cytosine (CC), those who have one cytosine and one thymine (CT). Trisomic people can be divided into four groups (CCC, CCT, CTT, TTT).

1.3 The microarray technology

SNPs require however a heavy process to be determined. If technical implementation is quite complicated, its principles are very simple. Its purpose is

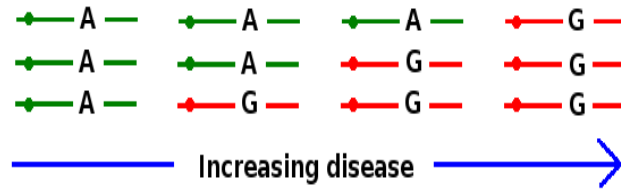


Figure 1.1: Idea of the analysis of SNP: a SNP can take 2 values (here A or G). A trisomic people have 3 chromosomes, hence 4 classes that are correlated to a disease.

to quantify the amount of a certain DNA sequence in a cell.

A DNA microarray is a collection of microscopic DNA spots attached to a solid surface, such as glass, plastic or silicon chip forming an array. The amount of DNA is amplified by a technic known as PCR and then denatured to obtain a single strand of DNA instead of two (Reverse Transcription). The affixed DNA segments are known as probes, thousands of which can be used in a single DNA microarray. This will allow those probes to bind to its complementary strand from the target, the target being the cell we want to study: the DNA present in the cell will be to the probes. Since we know the DNA sequence of the probes, we will know the DNA sequence a cell contains at a specific moment.

A cDNA molecule that contains a sequence complementary to one of the single-stranded probe sequences will hybridize, or stick, via base pairing (A with T, C with G) to the spot the complementary probes are affixed at. The spot will then fluoresce (or glow) when examined using a microarray scanner. Increased or decreased fluorescence intensity indicates that cells in the sample have recently transcribed, or ceased transcription, of a gene that contains the probed sequence ("recently," because cells tend to degrade DNAs soon after transcribing them). The intensity of the fluorescence is roughly proportional to the number of copies of a particular DNA sequence that were present and thus roughly indicates the activity or expression level of that gene. Arrays can paint a picture or "profile" of genes in the genome which are active in a particular cell type and under a particular condition.

One can distinguish two major types of applications concerning the microarray technology, the first one concerns transcriptomics and tries to find out which genes are differentially expressed. The second one deals with genomics and tries to find out the value of particular Single Nucleotide Polymorphisms. The first application can be of some use here because it also has to do with measures issued from microarrays and is provided with a detailed

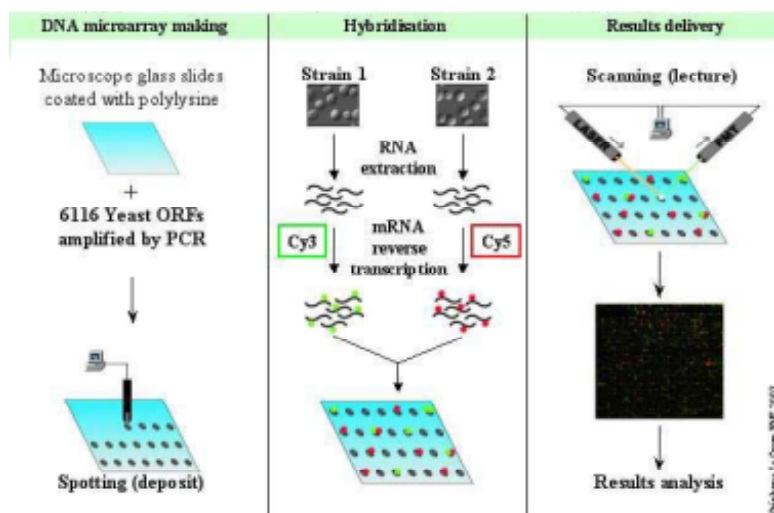


Figure 1.2: From <http://www.transcriptome.ens.fr/>: principles of Microarrays applied to cDNA. Instead of cDNA, we analyze here the two possible values of SNP by marking the DNA.

statistical documentation (See. [Huber],[Wilson],[Smyth], [Dudoit]).

For SNPs analysis, the DNA using a particular yeast is cut into small sequences, among which is the particular basis where the SNP is located. The sequence containing the SNP can have two (and only two) values. It can therefore hybridize with two different complementary probes. For example, let a SNP have two possible nucleotides C and G. If the sequence contains a C, it will bind with the probe that contains the cDNA of this sequence. If this probe is targeted with a molecule glowing green, the spot will glow green. If the sequence contains a nucleotide G, it will bind with a second probe. If this second probe is targeted with a red color, it will glow red (See graphics 1.3 page 8).

Therefore, in the disomic case (two chromosomes), if we observe only the red color, the patient is of class GG, if we observe only the green color, the patient is a class CC, if we observe both green and red color, the patient is of class CG. In theory, only three classes (green yellow, red) are possible for the disomic case and four (green, yellow, orange, red) for the trisomic case.

1.4 Presentation of the data

The data we study here are part of a protocol named "INTREPID" subsidized by the Jérôme Lejeune Institute (Paris). cDNA samples are analyzed at the Ecole Supérieure de Physique et de Chimie Industrielle and DNA samples

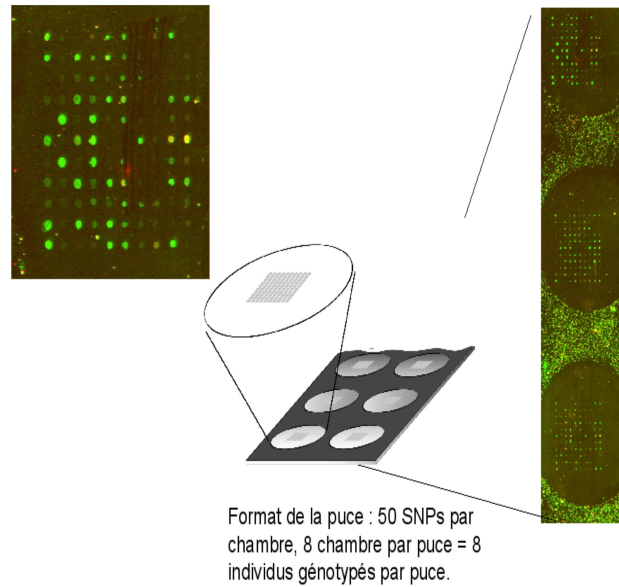


Figure 1.3: Arrays used in the experiment

are analyzed in Montpellier at the Human Genetic Institute (IGH in French), which gave a training data set to implement this analysis.

In our data set, 54 SNPs are analyzed, for both trisomic and disomic people (disomic people being the parents of the trisomic people). The conjoint analysis of parents and children should give further genetic information. Presently, we have 5724 observations for 54 SNPs, 53 people among who 31 disomic people and 22 trisomic people. For each people, we possess two spots. A spot is a place on the array referring to a SNP and a people where both a green and a red luminosity have been computed thanks to a scanner. Within one or two years, the number of people should be about several hundreds.

Among the SNPs, 50 are real SNPs and 4 are control spots to check the quality of the green and red channels. These 50 SNPs cover 2 Megabasis and 14 genes of the chromosome 21 (See figure 1.4 page 9). Coding and non-synonymous SNPs, which are located in genes supposed to influence trisomic phenotypes are analyzed. Non-coding SNPs are also analyzed for genetic reasons (linkage disequilibrium). The type of arrays used to process the computation of the green and red luminosity for the data training is four people per arrays, ie $4 * 54 * 2$ observations per arrays (See figure 1.3 page 8). A new type of array is yet used at the IGH.

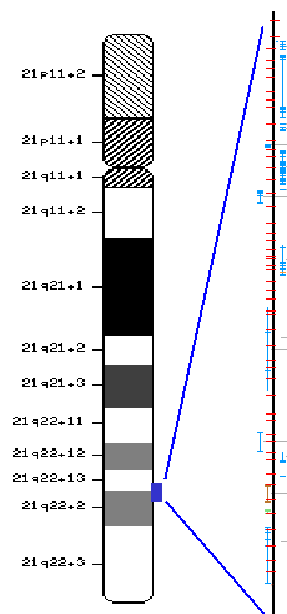


Figure 1.4: The chromosome 21: in red are the location of the SNPs and in blue are the coding regions.

Chapter 2

Normalization of the data

Microarray analysis technology is based on the idea that one can represent a particular basis in a DNA sequence (here the two possible values of the SNP) by measuring a light intensity which indicates the presence of a nucleotide. Two channels are used, a green one that indicates one of the value of the SNP and a red one that indicates the second possible value of the SNP. However, to compare luminosity level in both green and red channels, a number of transformations must be carried out on the data.

Several reasons can motive such a normalization step (See [[Quackenbush](#)]):

- unequal quantities of DNA after the PCR-RT process
- differences in labeling or detection efficiencies between the fluorescent dyes used
- systematic biases in the measured expression levels (variance bias)

Within normalization and between normalization have to be distinguished. Within normalization takes care to transform observations in a same array so as to make comparison between green and red channel relevant. Between normalization transforms observations in all arrays so as to make comparison between all the green and all the red channels relevant.

Three different types of normalization are described below. The LOESS normalization is a standard normalization used in cDNA experiment. However, for SNPs, this normalization turns out to have poor results. A second normalization method, called here "mean" normalization, uses the idea of the IGH. A third normalization, known as "quantile" normalization is also described. It provides good result and can be applied for both within and between normalization.

2.1 LOESS normalization

A common normalization is the so called LOESS procedure ([Wilson], [Quackenbush]). Data are first transformed in this way:

$$A = \log (\text{green} / \text{red})$$

$$M = \log (\text{green} + \text{red})$$

Plotting such a data is known in the cDNA literature as MA plot (See [Dudoit]). It allows to visualize bias induced by different sensibility. This bias is then corrected thanks to a loess procedure. Recall that loess procedure is a locally weighted regression in which fitting is done locally by weighted least squares. That is, for the fit at point x , the fit is made using points in a neighborhood of x , weighted by their distance from x . The size of the neighborhood is controlled by an α parameter (here set to 0.75) : it includes α % of the points, and these have tricubic weighting proportional to $(1 - (\frac{dist}{maxdist})^3)^3$ with the 'maximum distance' assumed to be $\alpha^{1/p}$ times the actual maximum distance for p explanatory variables.

However, this process is highly sensible to extrema values (more than 40% in practice. [Wilson]) and therefore we should be extremely precautionous when using this method for SNPs: in a SNP experiment, we are interested in comparing amount of green and red light. But for homozygote people (for example people with only C nucleotides on every chromosomes), green and red luminosity are very different. For example, green light will be high and red light close to null. As a result, every homozygote people should have a statistic $A = \log(\text{green}/\text{red})$ non close to 0 and should be therefore identified as an "extrema" value. Yet homozygote people can represent far more than 40% of the population.

To solve the problem of extreme values, we implement an algorithm:

1. produce a loess smoothing on all the data set, estimate the trend and subtract
2. compute the 40 % and 60 % quantiles
3. select values between those two quantiles
4. using this subset, redo step 1 until the quantile values do not change more that 5 %

The result is shown in figure 2.1 page 13. This algorithm is often used in the cDNA experiment and is known as composite normalization (See.

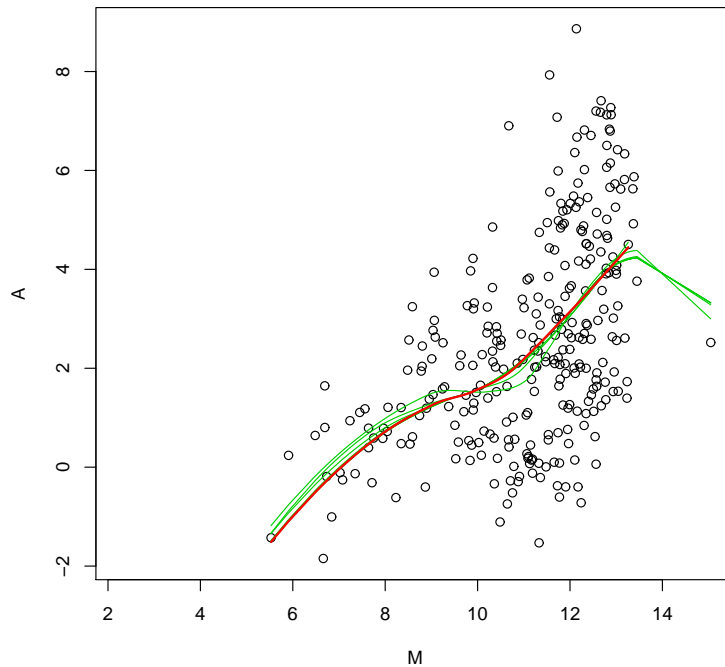


Figure 2.1: Estimation of the bias using loess method. The process have been iterated using a robust subset. The red line is the final estimation of the bias

[Smyth], [Dudoit], [Wilson]), that is to say using a robust subset to estimate an averaging trend and subtract this trend in all the data set.

Here is an example of the output produced by the R console when doing an automatic adjustment of the smoothing parameter. This parameter occurred in the last step of the algorithm:

```
Within normalization of array1
Call:
loess(formula = A ~ M, data = data[subset2, ])
```

```
Number of Observations: 108
Equivalent Number of Parameters: 4.78
Residual Standard Error: 0.4759
```

2.2 Mean normalization

Red and green probes do not have the same sensibility. In particular, we observe that green light has a range of values larger than the red light. This could be a possible damage for a further study if we want to compare green and red channels through the ratio $\frac{G}{G+R}$ and could lead to an underestimation of a certain class of SNP. A first idea is to develop the normalization method used at the IGH (See [Haquet]) where red observations are multiplied by a factor depending on the scale of the green values.

A first idea is to multiply red values so as to ensure that the red and the green means are equals. This idea is motivated by the fact that the distribution could look like an exponential distribution. Hence we use the following 'mean' normalization: $red = red * \frac{mean(green)}{mean(red)}$ However, we find that a Kolmogorov-test leads to too low values to justify an exponential distribution and that this normalization leads to outrageous maximal red values.

2.3 Quantile normalization

The goal of the quantile normalization procedure is to make two identical distributions as concerns their quantile values (See [Bolstad]). We applied this procedure on the green and red channels of each array in the SNP experiment. However, it can be also used to normalize probes between arrays in a cDNA experiment. This method is motivated by the fact that in a quantile-quantile plot, two random vectors have the same distribution if plotted data forms a diagonal line. If not, the two distributions are not the same. This suggests we can get two identical distributions by projecting values on the diagonal line.

This method is generalized to n vectors. n in the SNP case equals to 2 as we do normalize for each array green vs red channel. n equals to the number of arrays in the cDNA case as we do normalize the green or the red channel in all the arrays.

Let $q_k = (q_{1k}, \dots, q_{ik}, \dots, q_{nk})$ be the vector of the k^{th} quantile for the n^{th} vector and $d = (1/\sqrt{n}, \dots, 1/\sqrt{n}, \dots)$ be the unit vector. Project the data so as they all lie on the diagonal line using this projection of q onto d :

$$proj_d(q_k) = \left(\frac{1}{n} \sum_i q_i, \frac{1}{n} \sum_i q_i, \dots, \frac{1}{n} \sum_i q_i \right)$$

This implies that we can make two identical distributions by substituting the mean of the considered quantile to the original data over the n vectors.

2.3.1 First algorithm

This suggests the following algorithm to normalize data by making their quantiles equal. Note that this algorithm only applies to data with the same amount of observations (See [Smyth] and its library **limma**).

1. Make a matrix M with the data of n columns and p lines, p being the number of observations pro vector¹.
2. Sort each column of M . Let's call the new matrix M_{sort}
3. For each row, calculate the mean and assign it to each data in the row. Let's call the result M_{sort}' .
4. Get normalized data by rearranging M_{sort}' in the initial order.

This method is a specific case of the $F^{-1}(G(x))$ transformation where we estimate G by the empirical distribution of the n vectors and F using the empirical distribution of the average sampled quantile. This algorithm is employed in the within normalization where the green and the red channels for each array do have the same amount of observations.

2.3.2 Second algorithm

If the number of observations in each data is not the same, we propose another algorithm to the work. Each of the n vectors (which represent here the observations in each arrays) has p_i observations. Let p be the maximal value of the p_i .

1. For each vector sort the values to get X_i vector
2. Calculate the vector of quantile for each vector $Q_i = (0, \dots, \frac{i}{p_i}, \dots, 1)$. The k^{th} value of the Q_i quantile is given by the k^{th} of X_i .
3. For i in 1 to n , use spline interpolation to get from X_i and Q_i an approximation of the p quantile for each vector.
4. Calculate the mean of n of the p quantile.
5. Using spline interpolation, estimate the p_i quantile and replace its values in X_i
6. Reorder X_i in its initial order.

¹Note that this algorithm implies a same number of observations pro vector

This algorithm presents two major qualities from a programming point of view: first it allows different sizes of vector (and therefore can allow missing values) and can allow a smoothing estimate of the quantile in step 3 if we introduce a smoothing parameter in the spline method known in R as *spar* (The coefficient lambda of the integral of the squared second derivative in the fit (penalized log likelihood) criterion (See [Härdle]) is a monotone function of 'spar') which has been set here to 0.6.

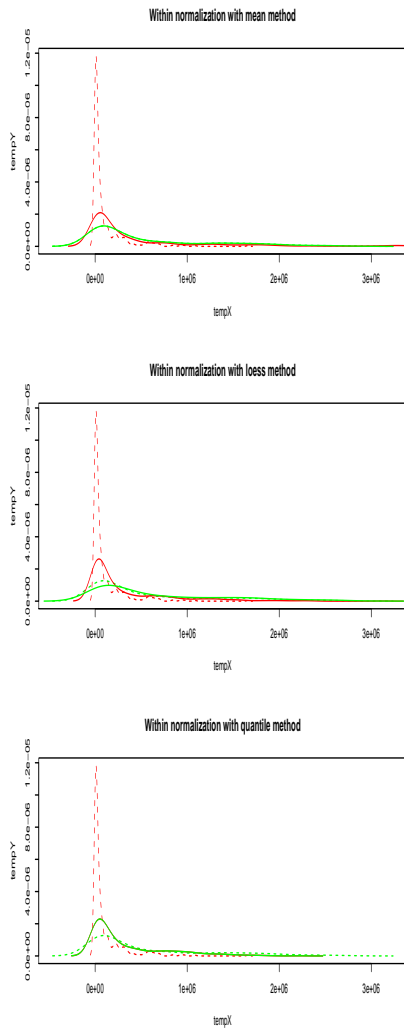


Figure 2.2: Within normalization of the first array. Raw data are in dashed lines and normalized data in plain line.

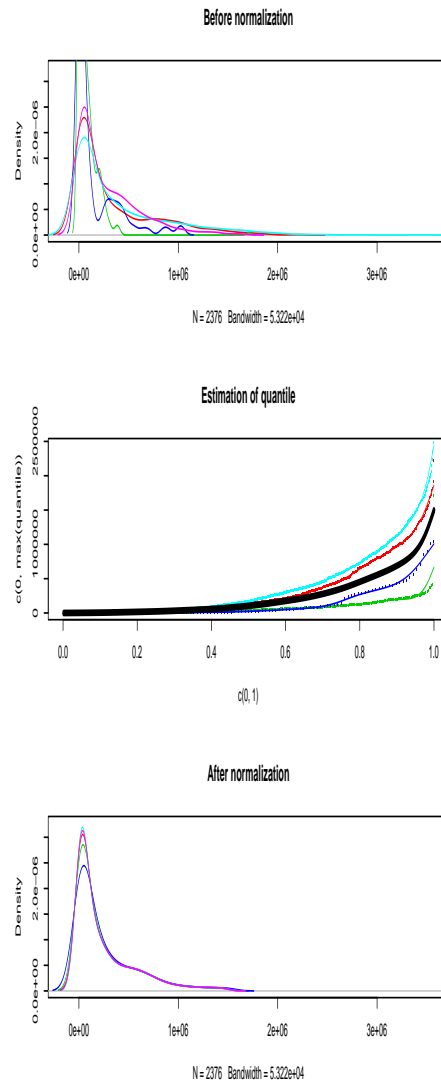


Figure 2.3: Between normalization for red channel using the second algorithm of the spline algorithm

Chapter 3

Classification of the data

Recall that the statistic we use to classify observations is the following: the statistic is defined by $x = \frac{green}{green+red}$ where *green* and *red* are the amount of green and red light computed by a scanner. This statistic is supposed to take at most three values for disomic people (0, 0.5, 1) and four for trisomic people (0, 0.33, 0.66, 1). It is possible that this statistic takes less than three or four values if some alleles are not present but values should theoretically be one of these.

Let μ_k^* be the expected value of the statistic for the k^{th} class. In practice, μ_k^* is always observed with a noise term ϵ : $\mu_k = \psi(\mu_k^*, \epsilon)$. In the following model, a gaussian mixture is assumed to model μ_k . Some additional constraints are added to f to fit the model to the needs of the experiment. Then an EM algorithm is implemented to find optimal parameters for this model. Three variants of the model are proposed. In a last part, criterions used to find the best model are presented.

3.1 Modeling the data

Note that the exact number of classes for a SNP is unknown. For example, in the third SNP, only two classes are apparently observed for disomic and trisomic people (no red and orange class). The number of classes is therefore a variable G and $max(G) = 3$ for disomic people and 4 for trisomic people. The log-likelihood for a mixture model with G components and n independent observations with same distribution is the following:

$$L(\tau_k, \theta_k | x) = \sum_{i=1}^n \log \left(\sum_{k=1}^G \tau_k f_i(x_k | k) \right)$$

where τ_k is the proportion of observations belonging to the k^{th} class, f_k its density and θ_k its parameters. x_k is the vector of observations of the i individual. Since observations are duplicated twice for each people, we have

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix}$$

If we assume that f_k is a bivariate gaussian density ϕ_k , θ is characterized by the expectation of class k , which should be close to its theoretic value μ_k^* , and the covariance matrix of class k , Σ_k . There is some motivations to assume a proportional covariance matrix among classes: in the following model, Σ_k is characterized by a covariance matrix C , where diagonal elements are identic, multiplied by a volume λ_k varying among classes. Indeed, noise is due to the difference between dyes and arrays (See [Wilson]) and is due also to the spacing which separates two duplicates (See [Smyth]). Since data have been normalized within and between arrays to remove bias and since this spacing is the same for every observations, a same structure for the covariance among classes is expected, we can assume that diagonal elements may be identic. Similar assumptions can be found in [Smyth]

These specifications can be summarized according to these constraints:

$$\begin{aligned} \sum \tau_k &= 1 \\ \mu_k &= \begin{pmatrix} \mu_{k1} \\ \mu_{k2} \end{pmatrix} \quad \text{with} \quad \mu_{k1} = \mu_{k2} \approx \mu_k^* \\ \Sigma_k &= \lambda_k \begin{pmatrix} \sigma & \omega \\ \omega & \sigma \end{pmatrix} \end{aligned}$$

As a result, we get the following density:

$$\phi_k(x_i | \mu_k, \lambda_k, C) = \frac{1}{2\pi\lambda\sqrt{\det(C)}} \exp^{-\frac{\lambda_k}{2}(x_k - \mu_k)'C^{-1}(x_k - \mu_k)}$$

The difficulty is to find a good estimation of these parameters.

3.2 The EM algorithm

The EM algorithm is a general approach to maximize a likelihood function with one random variable y_i recoverable from two other random variables x_i and z_i where x_i is observed and z_i is not observed. Here x_i is our matrix of observations for individual i and $z_i(k)$ is the probability that individual i belongs to a particular class.

If y is iid, with density ϕ and parameter θ , the complete data likelihood is (See [Fraley]):

$$L_C(y|\theta) = \prod_{i=1}^n \phi(y_i|\theta)$$

The observed data likelihood can be obtained by integrating z out of the complete data likelihood

$$L_O(x|\theta) = \int L_C(y|\theta) dz$$

The maximum likelihood estimate for θ based on the observed variable y maximizes L_O . Assuming a normal distribution, we get following complete data loglikelihood:

$$l(\mu_k, C_k, z_{ik}|y) = \sum_i^n \sum_k^G z_{ik} \log(\tau_k f_k(x_{i1}, x_{i1}|\mu_k, C_k))$$

The EM algorithm maximizes this loglikelihood by oscillating between two steps, the "Expectation" step and the "Maximization" step. In the E step, conditional expectation of the complete log-likelihood given the observed data and the parameter estimate is computed. The M step computes parameters that maximize the expected log likelihood from the E step values. EM can be shown to converge to a local maximum of the observed data likelihood (See [Titterington]). Although these conditions do not always hold in practice, the EM algorithm has been widely used for maximum likelihood estimation for mixture models with good results.

A tolerance parameter α is used to terminate the iteration of E and M step: $|L_O(\theta_n|y) - L_O(\theta_{n+1}|y)| < \alpha$. α is set by default to 1e-7.

```
# initialization (pas obligatoire juste pour le warning)
me <- mstepEEEsnp(data=data , z = em$z ,
equalPro = FALSE)

while (abs(loglik - em$loglik) > 1e-7) {
  loglik <- em$loglik
  me1 <- mstepEIE(data=data , z = em$z )
  if (me1$singular) {
    warning("Singular matrix in sep ",i , "\n\stopping alogrithm")
    break
  } else me <- me1
  em <- estepVVV(data=data,mu = me$mu, pro = me$pro ,
sigma = me$sigma)
```

```

    i = i + 1
    if (i > .snp$itmax[1]) break
}

```

3.2.1 Implementation of the algorithm

In the first E step, to initialize the EM algorithm, a hierarchical classification is implemented (See [Frale] and [Frale]). In this initialization, z_{ik} is defined as follows: $z_{ik} = 1$ if individual i belongs to class k 0 otherwise. In every other E step, z_{ik} is computed so as to maximize the a posteriori log-likelihood (See [Shashua])

$$\hat{z}_{ik} = \frac{\tau_i \phi_{k,i}}{\sum_{k=1}^G \tau_i \phi_{k,i}}$$

z_{ik} can be seen as the conditional probability that individual i belongs to class k . The maximum likelihood classification of individual i is therefore $j | z_{ij} = \max(z_{ik})$. The error probability can be easily computed through $p_i = 1 - \max(z_{ik})$.

The M step is the solution of a maximization program under constraints (See [Hahnloser] [Shashua]). With ϕ being the normal density, we get

$$\begin{aligned} \eta_k &\equiv \sum_i \hat{z}_{ik} \\ \hat{\tau}_k &= \frac{\eta_k}{n} \\ \hat{\mu}_k &= \frac{\sum_i z_{ik}(x_{i1} + x_{i2})}{2\eta_k} \end{aligned}$$

The estimation of λ_k and C is not straightforward and is found thanks to a second algorithm (See [Frale] [Celeux]). First compute

$$W_k = \sum_i \frac{z_{ik}}{2} (x_{i1}, x_{i2})(x_{i1}, x_{i2})' + (x_{i2}, x_{i1})(x_{i2}, x_{i1})'$$

W_k is thus defined instead $\sum_i z_{ik}(x_{i1}, x_{i2})(x_{i1}, x_{i2})'$ so as to make sure that diagonal elements in the covariance matrix are equal. We define also

$$W = \sum_k \sum_i \frac{z_{ik}}{2} (x_{i1}, x_{i2})(x_{i1}, x_{i2})' + (x_{i2}, x_{i1})(x_{i2}, x_{i1})'$$

One can show (See [Celeux] and [Celeux]) that $\hat{\lambda}$ and \hat{C} must minimize $\sum_k \text{tr}(W_k C^{-1}) / \lambda_k + 2 \sum_k \eta_k \ln(\lambda_k)$. [Celeux] proposes following algorithm:

$$\begin{array}{ll}
 \textit{Initial step} & \\
 \lambda_k = & (1, \dots, 1) \\
 C = & W \\
 \textit{Step1}(C \textit{ fixed}) & \\
 \lambda_k = & \frac{\text{tr}(W_k C - 1)}{2\eta_k} \\
 \textit{Step2}(\lambda \textit{ fixed}) & \\
 C = & \frac{\sum_k \frac{1}{\lambda_k} W_k}{\sqrt{\det(\sum_k \frac{1}{\lambda_k} W_k)}}
 \end{array}$$

```

while ( any(abs(comp1 - lambda) > .snp$tol[2] ) &
any(abs(comp2 - c) > .snp$tol[2] ) ) {
comp1 = lambda
comp2 = c

#first step C fixed, find lambda
for (k in nclass){
lk = w[, ,k] %*% solve(c)
lambda[k] <- sum(lk[c(1,4)]) / (2 * nk[k])
}

# second step lambda fixed, find C
num <- array(data=0,dim = dim(c))
for (k in nclass) num <- (1/lambda[k]) * w[, ,k] + num
c = num / sqrt(det(num))

i = i + 1
if (i > .snp$itmax[2]) {
warning("Computation of C and lambda did not converge")
break
}
}

```

3.2.2 Various models

Step 1 and 2 are done iteratively until $|\lambda_n - \lambda_{n+1}| < \beta$ and $|C_n - C_{n+1}| < \beta$, β being a tolerance parameter set by default to $5e-8$. One limit of the EM algorithm is that the computation of $\lambda_k C$ degenerates easily to singular matrix. In particular, if there is too few observations in a class (See [Fraleay]),

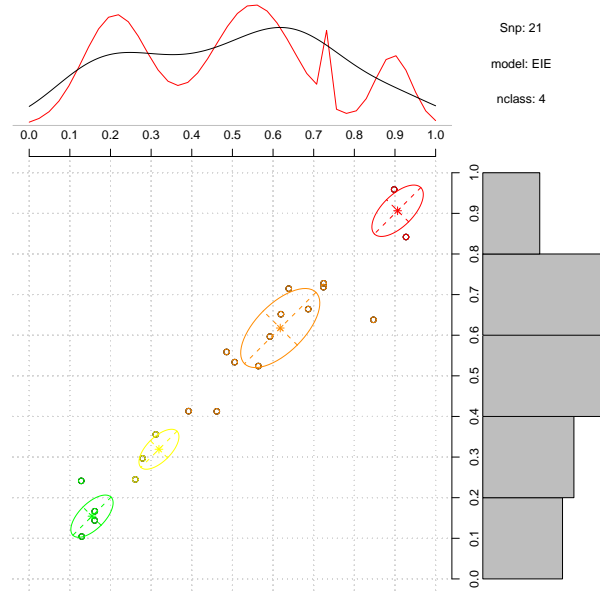


Figure 3.1: Figure showing the result of the EM algorithm for the 21 SNP. All classes have a proportional covariance matrix. Yellow and Orange classes have their means close to theoretic values 0.33 and 0.66. Green and Red classes are always being close to 0.2 and 0.9 than 0 and 1. This can be a result of background noise. Note that all the classes have a same shape: only the volume is different whereas the orientation of the shape is the same.

the computation leads to too small coefficients λ_k . To prevent singularity matrix in the computation, the following rule is used:

```
if (max(lambda) / Min(lambda) > threshold ) break
```

In practice, default value for threshold is set to 5. Note that this value can be changed (like α and β). When the threshold is reached, the last value of the parameters are given as final result and a warning is emitted.

The final result in this case may be compared with two other models. The first one takes $\Sigma = \Sigma_k = \lambda C$ as covariance matrix. Σ is estimated in the M step with W/n . The second one is a EM algorithm working in one dimension where each observation $x_i = (x_{i1}, x_{i2})$ is replaced by $x_i = 1/2(x_{i1} + x_{i2})$.

According to the literature (See [Fraleley], [Fraleley]), the $\Sigma_k = \lambda_k C$ model is called **EIE**, $\Sigma_k = \lambda C$ model **EEE** and the 1 dimension model **E**. **E** means *equal* and **I** *inequal*. **E** means equal covariance matrix in a one dimension model. **EEE** means equal orientation, equal volume and equal diagonal elements. **EIE** means equal orientation, inequal volume and equal diagonal elements.

3.3 Complements to the EM algorithm

3.3.1 Choosing the best model

Two different criterions BIC and ILC (See [Celeux]) can be used to choose the best model \widehat{m} and the number of classes \widehat{K} . The Bayesian Information Criterion minimizes the integrated likelihood over the possible number of classes and models

$$\log f(y|m, K) = \int_{\Theta_{mK}} \log f(y|m, K, \theta) \pi(\theta|m, K) d\theta$$

Schwarz has proposed, under regular conditions, this approximation where v is the number of free parameters:

$$\log f(y|m, K) \approx \log f(y|m, K\widehat{\theta}) - \frac{v_{mK} \log(n)}{2}$$

It leads to minimize the so-called BIC criterion with L_{mK} , the likelihood estimate with the EM algorithm in model m with K classes:

$$BIC_{mK} = -2L_{mK} - 2v_{mK} \log(n)$$

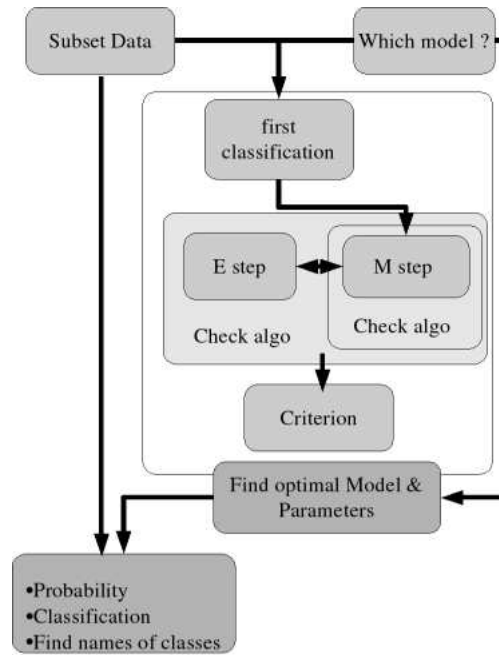


Figure 3.2: Sketch of the algorithm implemented in the statistical software.

Another criterion proposed by [Celeux] is the Integrated Likelihood Criterion. Whereas the BIC criterion does not take into account a cluster structure in the data, the ILC introduces the integrated likelihood of the complete data

$$\log f(y, z|m, K) \approx \log f(y, z|m, K\hat{\theta}) - \frac{v_{mK} \log(n)}{2}$$

This likelihood can be approximated using the BIC criterion. Assuming that $z \approx \hat{z}$, where \hat{t}_{ik} is an indicatrice variable resulting of the EM algorithm taking 1 as value if individual i belongs to class k , 0 otherwise.

$$ILC_{mK} = -2BIC_{mK} - 2 \sum_i \sum_k \hat{t}_{ik} \log(z_{ik})$$

Here is an example of ILC criterion for the third SNP. In practice, ILC should be preferred to BIC to find an optimal number of classes.

```

> bonf = makeSnp(trisoRaw,mesRes,3,plot=TRUE, +
+ flag2D = 4 , model = "EIE")
ILC for EIE with 1 classes equals to -67.56624
ILC for EIE with 2 classes equals to -4220.001
ILC for EIE with 3 classes equals to -3508.474
ILC for EIE with 4 classes equals to -3964.589
  
```

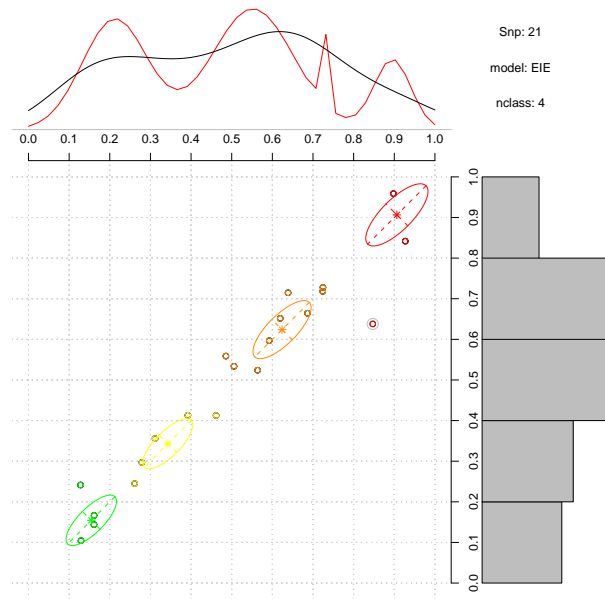


Figure 3.3: 21 SNP with subset. The red spot surrounded by a gray circle is not reliable enough to be used in the computation of optimal parameters. The resulting classification, compared with figure 3.1 page 24, shows a nearly similar covariance matrix among classes..

3.3.2 Subsetting data

[Quackenbush] warns against using poor quality spots in the analysis. Two criteria can be used to detect such spots:

```
if log(green) + log(red) < threshold
```

and

```
if covariance(yi) > threshold * covariance(y)
```

The first one says that in our model, at least one channel either green or red must give a minimal luminosity since there is no class of people for which both green and red light can be off. The second finds observations with too low correlation.

Those data can be retrieved from the EM algorithm. Once $\hat{\theta}$ has been found, their a posteriori probability z_{ik} are computed. In practice, subsetting data have a great impact on the shape of the classes and can lead to very different classifications. Such an impact may be assumed to diminish with the increasing number of observations.

Chapter 4

Clustering software

Two operator systems are possible to develop the software: the Windows environment and the Unix environment (more particularly Linux). Those two environments offer a bunch of languages and softwares. We have chosen the Unix system for many reasons:

- Practical reasons: informatic computations are far more easier with well-known and free languages such as R, and the Qt library for the C++ interface. Powerful software such as Kate, Kdevelop or RKward are also very helpful but use both the KDE desktop environments available only under Unix.
- Financial reasons: under Linux, you can have those softwares for free.
- Acknowledgement: all this work could not happened without the Bioconductor project, contributed packages or free publications on Internet. To work under the GPL license and under Linux is a way to thank those projects and to acknowledge the views of their founding fathers.

The software should be both robust and comfortable: therefore, the implementation has been divided in two parts according to those two main goals. The first part of the chapter deals with the core functions (statistical computations, experiment adapted tools). The second part is the implementation of the Graphical User Interface in C++, XML and bribes of php.

As a matter of fact, the resulting software can be fully used only under a Unix computer which runs KDE. However, if you do not have KDE but Linux, you can use the core functions as a R source file. You will not be able to benefit from the Graphical User Interface but the statistical program is available with HTML help files..

4.1 Statistical computation

4.1.1 Programming with R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories by John Chambers and colleagues. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible, which are the reasons for the choice of the R language (See [R]). The R language is used in the present work for all statistical computations.

Here is showed a two step algorithm for the M step in the EIE model:

```
"mstepEIE" <- function(data,z){
  [...]
  for (k in nclass){
    wk <- cbind(data[,1] - mu [1,k], data[,2] - mu [2,k])
    ck <- rbind(z[,k],z[,k],z[,k],z[,k])
    wk = apply(wk,1,fwk)
    wk = wk * ck
    wk = matrix(data=apply(wk ,1, sum ) , 2,2)
    wk[c(1,4)] <- mean(wk[c(1,4)])
    w[, ,k] <- wk
    c = c + wk
  }
  c = c/n
  lambda <- rep(1, max(nclass))
  nk = apply(z,2,sum)
  i = 0
  # first step C fixed find lemnda_k
  comp1 = rep(0,max(nclass))
  comp2 = matrix(0,ncol= 2, nrow= 2)
  while ( any(abs(comp1 - lambda) > .snp$tol[2] )
  & any(abs(comp2 - c) > .snp$tol[2] )) {
    comp1 = lambda
    comp2 = c
    for (k in nclass){
      lk = w[, ,k] %*% solve(c)
      lambda[k] <- sum(lk[c(1,4)]) / (2 * nk[k])
    }
  }
}
```

```

# second step lambda fixed, find C
# sum 1/lk wk / det(1/lk wk)^1/2
num <- array(data=0,dim = dim(c))
for (k in nclass) num <- (1/lambda[k]) * w[:,k] + num
c = num / sqrt(det(num))
i = i +1
if (i > .snp$itmax[2]) {
warning("Computation of C and lambda did not converge")
break
}
}
# looking for singular value
if (max(lambda) / min(lambda) > .snp$eps ) {
singular <- TRUE
} else singular <- FALSE
[...]
}

```

4.1.2 The 'snp' package

The program to classify the data according to their SNPs is available as a package named 'SNP' under Unix and as a workspace for Windows. It requires two other packages: *splines* (See [R]) used for normalization and *mclust* (See [Fraley]) for some functions used in the EM algorithm. Thanks to the package facilities, each function has been implemented with a corresponding help file (*.html and *.rd files.). Those help files contain also examples.

The 'snp' package is constructed around a class orientated language. Three major classes of objects have been endowed with specific functions to make the software more comfortable when using it under Windows. Those three classes are the classes *snpLayout* which is used to handle basic information about the experiment and to read the input, *snpRaw* where data are stored and normalized, and *snpRes* where the classification for each SNP is stored.

Here, you can find all the available functions to use the software :

Information on package 'snp'

Description:

Package: snp
 Title: find SNP for disomic people

```

Version:      2.1.1
Date:        2005-13-04
Author:      Adrien du Moulinet d'Hardemare
Description:  Normalize data and implement an EM
              algorithm to classify SNPs
Maintainer:  adrien d'hardemare
              <adrien.dhardemare@gmail.com>
Depends:     R (>= 2.0), mclust, splines
License:     GPL
Built:      R 2.1.0; ; 2005-05-27 14:54:44; unix

```

Index:

createLayout	Informations about the design
disoLayout	class disoLayout
emSnp	Internal function used to call the appropriate maximization algorithm
estepEEEsnp	intern function
interfaceSnp	Provide a simple interface to manipulate data directly
layout2Raw	layout2Raw
makeSnp	Calcul the model used to classify data
mstepEEEsnp	mstep for SNP
normBetween	Between array normalization
normSnp	Normalization of data
normalizeLimma	Quantile normalization
optionSnp	General options
selType	Select a type of data
snpRaw	snpRaw
snpRes	SnpRes
snpSnp	Class to access data
writeRes	Write classification in a table

Please note that the package may handle disomic **and** trisomic observations. Actually disomic and trisomic observations can be grouped on a same array. These observations will be normalized altogether and split afterwards according to their type using the *selType* function. A more detailed User Guide to use the 'snp' package can be found in the appendix (See page 49).

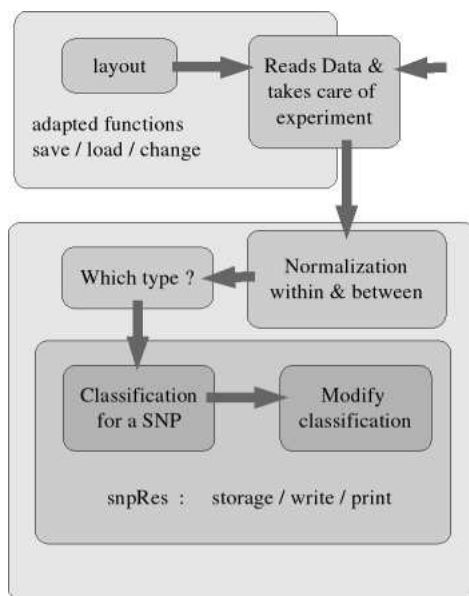


Figure 4.1: How the three main classes interact. Note that facilities have also been implemented to update classification when new observations are available

4.2 The Graphical User Interface

4.2.1 What is RKward

To use the statistical program in a friendly way, a Graphical User Interface has been implemented with RKward. It can be freely downloaded at www.rkward.sourceforge.net. RKward is meant to become an easy to use, transparent front-end to the R-language, a very powerful, yet hard-to-get-into scripting-language with a strong focus on statistic functions. Instead of developing an independant GUI (for example in TCL/TK as it is usual in Bioconductor), the GUI has been integrated in a software. This software RKward should not only provide a convenient user-interface but also take care of seamless integration with an office-suite.

RKward then is a project to propose a free replacement for commercial statistical packages (a kind of free SPSS for Unix system). It will be a transparent interface to the underlying R-language. That is, it will not hide the powerful syntax, but merely provides a convenient way, in which both newbies and R-experts can accomplish most of their tasks because a GUI can never provide an interface to the whole power of a language like R. In some cases users will want to tweak some functions to their particular needs and

to automate some tasks. By making the "inner workings" visible to the user, RKward will make it easy for the user to see where and how to use R-syntax to accomplish their goals.

At the present moment, a lot of work has been done. When reaching the project, RKward was in the 0.3.0 version. It is now in 0.3.2. Apart from one major problem (communication between Qt and R as concerns the threads that the console emulation send to R), we can hope a soon next release. Plotting and the management of the R-errors have been tackled. Otherwise the interface causes no particular problem and just needs time to be implemented. For the need of the present paper, numerous patches have been proposed to allow a full GUI adapted to the analysis of SNPs.

4.2.2 Architecture of a RKplugin

The user interface is a mix of four programming languages that allows every user to generate automatically R code through a widget (See figure 4.2 page 35) :

- the C++ backend which actually constructs the widgets and manages communication between RKward and R
- the XML interface, linked to the C++ backend allows a flexible use of this C++ part, even for people non familiar with C++
- the php backend : this client/server language allows the plugin to manage the interactivity between the widget and the user
- the R language itself which executes the statistical computations.

The C++ makes use of the free version of the Qt library to implement widgets. The improvements that have been proposed for RKsnp - apart from the work already done by Thomas Friedrichmeyer and Pierre Ecochard and from the debugging work - permit to construct a browser widget and an input widget. We also made patches to add features (make widgets linked together, extended selection of variables, management of the size and other numerous patches). The output windows, the help function and the windows system are not yet fully implemented.

Here is an example of code (a mere input widget that allows the user to enter any value in a text editor lines). This is for example used in the plugin which aims at adding array to a previous layout (See figure 4.3 page 36) :

```
#ifndef RKINPUT_H
#define RKINPUT_H
```

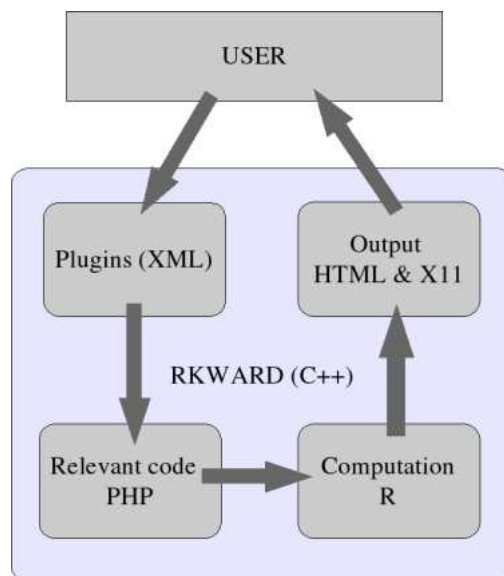


Figure 4.2: How Graphical User Interfaces are implemented in RKward

```

#include <rkpluginwidget.h>
class QTextEdit ;
class QLabel ;
class QVBoxLayout;

/**
 * @author Adrien d'Hardemare
 */
class RKInput : public RKPluginWidget
{
    Q_OBJECT
public:
    RKInput(const QDomElement &element,
            QWidget *parent, RKPlugin *plugin);
    ~RKInput();
    void setEnabled(bool);
    QVBoxLayout *vbox ;
private:
    QString depend;
    QTextEdit * textedit ;

```

```

QLabel * label ;
QString size ;
public slots:
void slotActive();
void slotActive(bool);
void textChanged();
protected:
/** Returns the value of the currently selected option. */
QString value (const QString &modifier);
};

#endif

```

Once the C++ backend is implemented, the way RKward is built (will) allows you to forget C++ entirely and to create complex widgets through a more friendly language, namely the XML. XML is a tag language: to each tag corresponds a widget with particular features defined in the C++ backend.

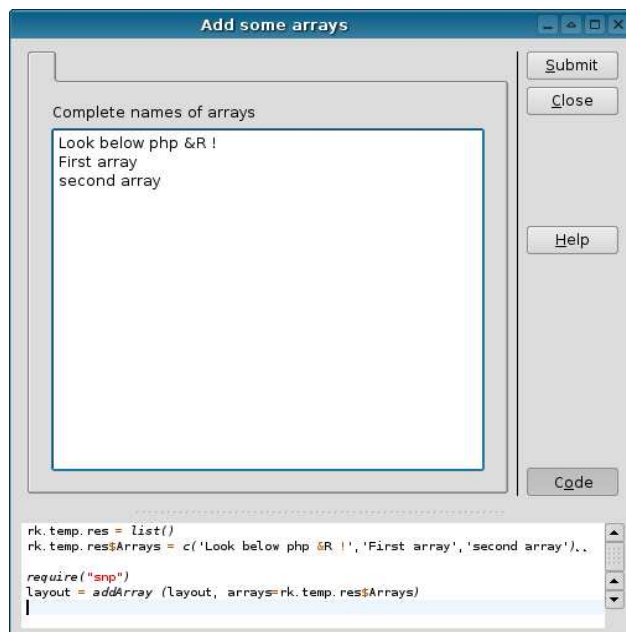


Figure 4.3: Simple example of plugin. (Here to add an array to a previous layout). The R code is visible in the lowest part of the plugin.

Example of XML code to create a simple widget which aims at adding array to a previous layout (See figure 4.3 page 36). In following code, a

parent widget named "Add some arrays" of type dialog is created. An input widget has been created within a tab:

```
<!DOCTYPE rkplugin>
<!-- This is a simple example, of how a "plugin"
might be configured. -->
<document>
  <entry type="entry" id="addarray" label="Add some arrays" />
    <dialog>
      <tabbook>
        <tab>
          <input size="big" id="arrays"
            label="Complete names of arrays" />
        </tab>
      </tabbook>
    </dialog>
</document>
```

This XML code is only the visible part of the widget. It is stocked in a file called `description.xml`. To this file is added another one named `code.php`. In this second file is stored the R code the widget is supposed to execute. To make this R code flexible to the user's will, it is mixed with php code. The php inquires about the user's input and gives it to the R code.

Here we see a php function called *calculate* that RKward executes when the user presses *submit*. The php code is between tags `<? and ?>`. Outside those tags, we have the R code of the `snp` package. Apart from a *calculate* function, a *printout* and *clean* function must be also implemented for each plugin to print out output in a HTML way and to clean the workspace. Here for the same plugin "add some arrays", we get :

```
function calculate () {
$vars =  str_replace ("\n", "','",
  trim (getRK_val ("arrays")))  ;
?>
rk.temp.res = list()
rk.temp.res$Arrays = c('<? echo $vars; ?>')

require("snp")
layout = addArray (layout, arrays=rk.temp.res$Arrays)
<?
}
```

4.2.3 Presentation of 'RKsnp'

Put the folder names RKsnp in the folder where are stored your plugins for RKward (you can find it by looking the settings). A menu named *RKward* should appear in the entry *Analyze*. A list of several plugins pop-ups when clicking on *RKward* should appear grouped in four folders: input, analysis, update, output. They contain the different possible plugins of RKsnp. They are stored in the order of the analysis.

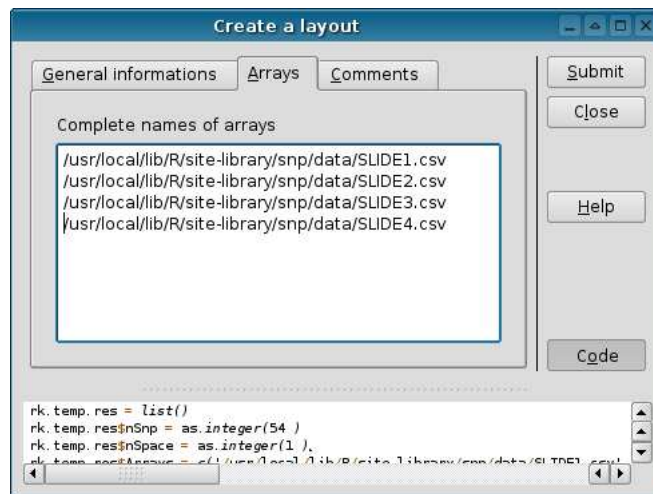


Figure 4.4: Create a layout. In the "Arrays" tab, you must give the complete names of the file where are stored your informations. In those files should be stored data with same experiment conditions.

In the first entry named *layout* are stored several plugins that allow you to deal with the way you have stored your information on your experiment. (See figure 4.4). Note that if you want to save or load a previous work, you just need to save your workspace in the file menu. Please note that the software can only take into account experiment where observations have been duplicated twice. A further development should make this problem solved. In the spacing option, you must give the number of observations which separate duplicated observations in your files. The tab *format* allows you to use any format to store your observations. If a problem occurs during

the input reading, a warning will appear. In this plugin, you must also specify the number of the columns where needed informations are stored.

The analysis can begin. In a first step, we will normalize data using appropriate plugin. Note that produced plot will look like figures 2.3 and 2.2 page 17. For the within normalization you can choose between 'mean' method and 'quantile' method. The 'loess' method described in chapter 2 is not available since it provides poorer results. The 'spar' is to be used only by advanced user (See page 16). It allows you to manage the smoothing parameter in the spline normalization in between normalization. After normalization, you must then select a type of people you want to analyze (either disomic people or trisomic people) using the plugin *select a type*.

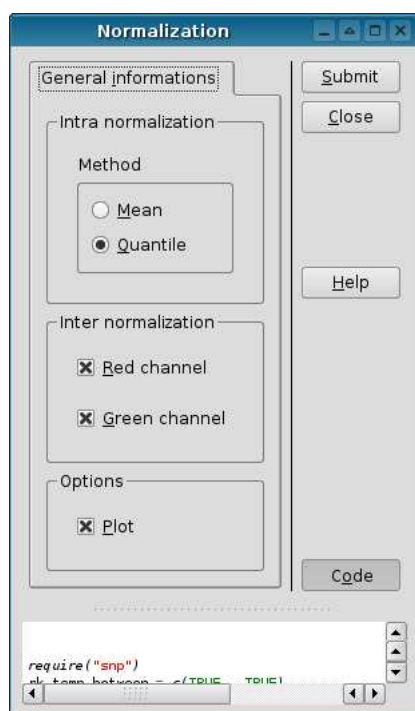


Figure 4.5: Normalize your data

When you click on several possible classes and models, a criterion will be used so as to find the best model. Two criteria are currently available: the ILC and BIC criterion. Note however, that the BIC criterion is a conservative criterion which leads to select only few classes. We recommend therefore the ILC criterion. You can also select a subset of 'good quality' spots by adjusting the minimal luminosity and the maximal covariance. Flagging spots will be removed of the computation of optimal parameters but will be classed once

these parameters are found. In the 'Options' tab, (only for advanced user), you can specify the maximal number of iterations, the scale parameter which avoids degenerating EM algorithm (larger scale parameter will lead to classes with more different variance) and tolerance parameters which tell RKward when it should finish the computations. You can also modify the interval of the colors which are used to automatically find the name of the found classes.

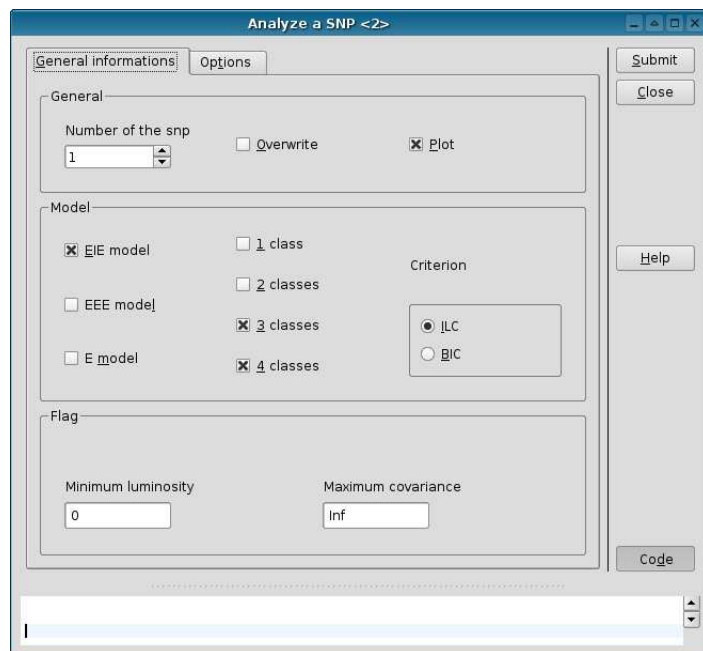


Figure 4.6: Analyze a particular SNP. Note, that if you only want to try to a particular classification and you do not wish to keep the result, you can use the plugin *Test/write* to tell RKward when it should store results.

An update folder indicates you that you may add new data to a previous classification. If you wish it, classification will be updated. This classification assumes that in a previous work, you have already classified SNPs and that the correct model and number of classes have been found. The plugin will ask you to give a file where those information are stored. This file can be automatically generated using the plugin *Information on SNP* in the output folder.

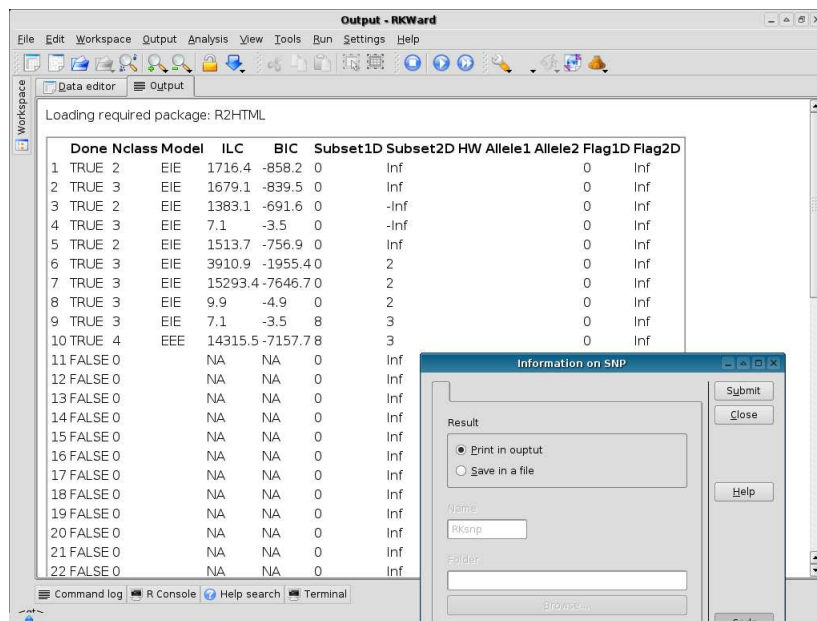


Figure 4.7: Information about SNP can be viewed thanks to appropriate plugin. In a same way, classification can be viewed.

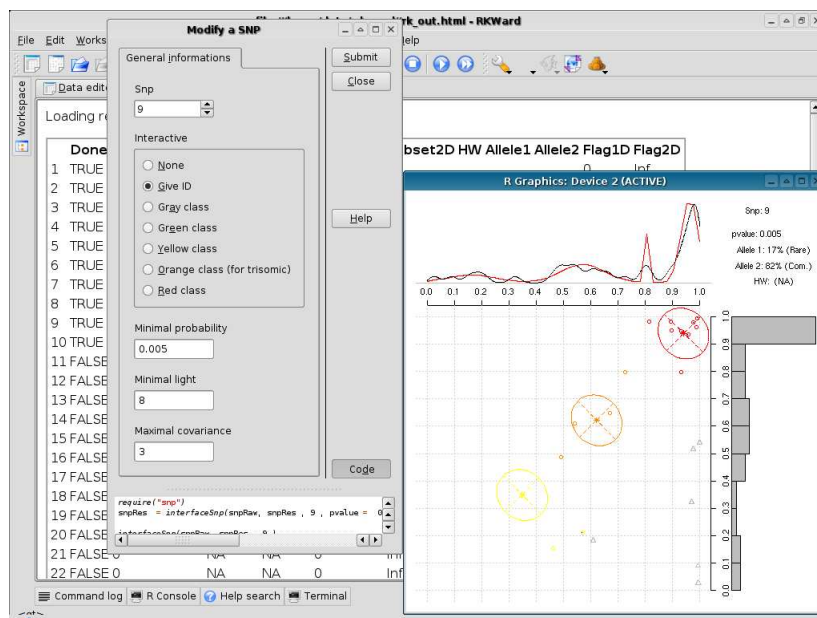


Figure 4.8: Modify the results of a SNP: observation with too low luminosity and too high correlation have been flagged (gray triangle). Left click on the mouse gives the ID of people. Right click on the mouse finishes the ID mode.

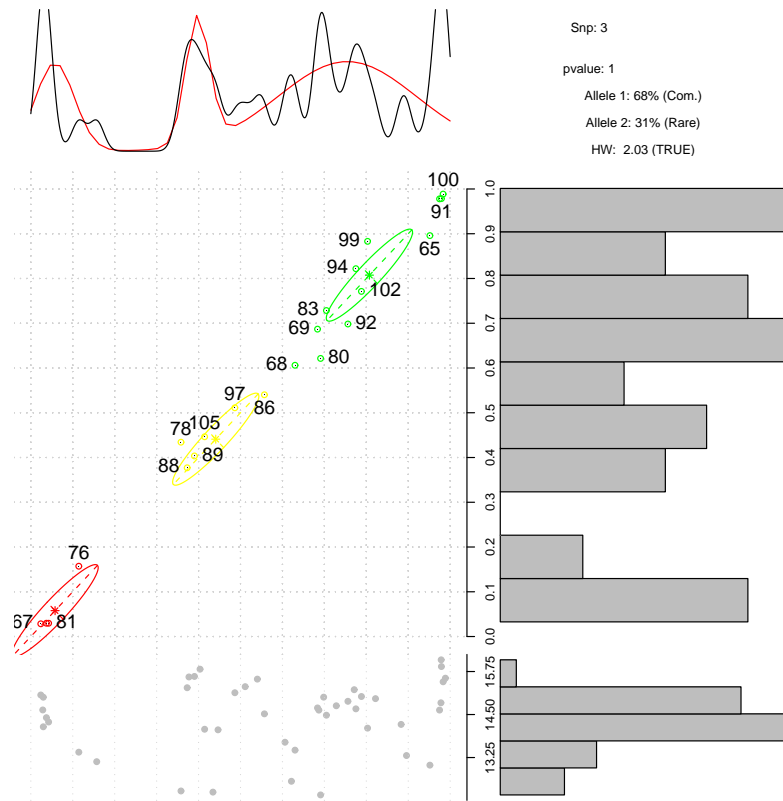


Figure 4.9: Note a new type of graphics (06/02/05) which gives more informations on particular SNP. Beyond the main graphics, a new plot had been added to give the total luminosity for each duplicates. It can turn out to be very useful to point out spots with too low luminosity.

Once your SNP has been analyzed, you can modify result. Note that if you change a particular spot of class, its p-value will be removed and it will be marked with a triangle on the graphic. You can put observations in 5 classes: red, green, yellow and orange (only for trisomic people). The gray class regroups all observations that failed to be classed with efficiency. You can also select in this plugin the desired p-value for the classification.

Conclusion

At the current state of development, the data training set has allowed the implementation of a classification software applied to microarray data. The classification is based on a robust statistical analysis, which gives good results provided a minimum quality of the data. Debugging the software and improving the RKward interface are presently the two main remaining works.

Of course, a lot of improvements can be done so as to make the software more robust and more comfortable. However, apart from those general improvements, we can list some improvements which should make the software more powerful.

1. Some informations stored in the scanner result files (background noise, volume of the spots) are not yet taken into account. Perhaps, they could lead to a better normalization or define a better subset in the EM algorithm.
2. The models used in the algorithm should also be generalized to n duplicates with an adequate covariance matrix. At the present moment, only 1 or 2 duplicates per people is allowed.
3. Sequential problems must be taken into account. In such an experiment, all observations are not obtained in a simultaneous experiment but arrive spaced in time. The software should be able to handle this problem without obliging the user to redo all the classification for all the SNPs. Two solutions could be possible (See [\[Titterington\]](#)): assuming that the number of classes set in previous analysis is fixed, one could either update the EM algorithm or tries with prior information on parameters a classification of new data using a bayesian framework. At the present moment, the first solution has been implemented without any further development.

To help with the criticism of results and giving another point of view to assess the quality of the classification, we have also implemented a kernel estimation of the distribution of the spot between the range 0 and 1. Two

density estimations are proposed : a non parametric estimation which adjusts its bandwidth so as to minimize the unbiased cross validation criterion proposed in the R function *bw.nrd* (See [Fraley], [Fraley] and [Härdle]) and a semi parametric estimation which assumes a gaussian mixture with known number of classes. This density can be viewed in figure A.2 page 57 and figure 3.1 page 24.

To help interpreting the results, two types should also be implemented. The first one, based on a chi-square test, should test whether a certain genetic property known as the Hardy-Weinberg equilibrium can be hold for disomic people. The Hardy-Weinberg equilibrium answers this question: is the proportion of AA equal to npp , the proportion AB to nqp and the proportion BB to nqq ? Further information should be also retrieved from alleles. For example if an allele is rare or common (inferior or superior to $\sqrt{(3/n)}$).

Further improvements should also be implemented to correlate genomes of people characterized by SNPs and their phenotype. For example, a factorial analysis could be implemented, taking the values of the SNPs as explanatory variables and projecting phenotype features on the result. One could also implement a test to see whether proportions of alleles differ in particular subset defined by phenotypic character.

Bibliography

- [Baptista] Baptista et al., Hum; Genet., 2000
- [Benjamini] Y. Benjamini, Y. Hochberg. 1995, Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series*
- [Bolstad] B. M. Bolstad ,R. A. Irizarry, M. Astrand and T. P. Speed , 2002 , A comparison of normalization methods for high density oligonucleotide array data based on variance and bias, *Bioinformatics*
- [Carreira] Miguel A. Carreira-Perpinan, Christopher K. I. Williams, On the number of modes of a gaussian mixture www.cs.toronto.edu/~miguel/papers/ps/sst03.pdf
- [Celeux] Celeux G and Govaert G, 1995, Comparison of the mixture and the classification maximum likelihood in cluster analysis. *Journal of statistical computation and simulation*
- [Celeux] Celeux G and Govaert G, 2002, Mixmod, Statistical documentation. www-math.univ-fcomte.fr/mixmod/statdoc/statDoc.html
- [Dudoit] Sandrine Dudoit, Juliet P. Shaffer, Jennifer C. Boldrick, 2003, Multiple Hypothesis Testing in Microarray Experiment, *Statistical Science*
- [Dudoit] S. Dudoit, Y. H. Yang, M. J. and C. T. Speed, 2000, Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiment. Technical report, *Stanford University School of Medicine*.
- [Fraley] Chris Fraley and Adrien E. Raftery, 2002, Model Based Clustering, Discriminant analysis and Density estimation. *Journal of the American statistical association*.

- [Fraley] Chris Fraley and Adrien E. Raftery, 2002, MClust : Software for Model Based Clustering, Discriminant analysis and Density estimation. User Guide, *University of Washington*
- [Hahnloser] Richard Hahnloser, 2004, Course on Biophysics of neural system, working paper, *University and ETH Zurich*
- [Haquet] Haquet, E., Laurent, AM., Buard, J. 2003, TRISOMIE 21, SNPs ET PUCES : Trisomie 21, SNPs et puces : variabilité phénotypique du syndrome de Down., Presentation, *IGH*
- [Hardemare] Adrien d'Hardemare, 2004, Introduction à l'étude statistiques des microarrays, working paper, working paper, *ENSAE*
- [Härdle] W. Härdle, M. Müller, S. Sperlich, A. Werwatz, 2004, Nonparametric and Semiparametric Models, *Springer-Verlag*.
- [Huber] W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, M. Vingron, 2002, Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*.
- [Quackenbush] John Quackenbush, 2002, Microarray data normalization and transformation, *Nature Genetic Supplement*
- [Nichols] Thomas Nichols, Satoru Hayasaka, 2003, Controlling the Family Error Rate in functional neuroimaging : a comparative review. *Statistical Methods in Medical Research*
- [R] R Development Core Team (2005). R: A language and environment for statistical computing. *R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, <http://www.R-project.org>.*
- [Shashua] Amnon Shashua, 2004, Introduction to machine learning, Soft Clustering, K-means, Gaussian mixture, working paper.
- [Smyth] Gordon K. Smyth, Joëlle Michaud, Hamish Scott, 2003, Use of within-array replicate spots for assessing differential expression in microarray experiments, *Bioinformatics*.
- [Smyth] Gordon K. Smyth, 2004, Linear models and empirical bayesian methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*

- [Titterington] D. M. Titterington, A. F. M. Smith, U. E. Makov, 1985, Statistical analysis of finite mixture distributions, *Wiley Series in Probability and Mathematical Statistics*
- [Venables] Venables, W. N. and Ripley, B. D., 2002, Modern Applied Statistics with S. *Springer*.
- [Wilson] D. L. Wilson, M. J. Buckley, C. A. Helliwell, I. W. Wilson, 2003, New normalization methods for cDNA microarray data calibration and to the quantification of differential expression, *Bioinformatics*.
- [Yang] Y. H. Yang, S. Dudoit, P. Luu, D. M. Lin, V. Peng, J. Ngai, T. Speed, 2002, Normalization for cDNA microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Res*

Appendix A

User Guide for the R Package

A.1 Installation of the package

The package requires two other packages: "mclust" and "splines". For the Unix system, type `sudo R CMD INSTALL snp.tar.gz` to profit from the full package utilities. Within the R console, type `library("snp")`.

```
> library(snp)
Loading required package: mclust
```

```
Attaching package: 'mclust'
```

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

```
density
```

```
Loading required package: splines
```

For the Windows system, load `snp.Rdata` file within the R console and then type `snp()` to initialize the program.

```
> load("diplomsarbeit/diso/pretpourigh/source/snp.RData")
> snp()
Loading required package: splines
Loading required package: mclust
```

```
Attaching package: 'mclust'
```

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

density

```
[1] "Everything seems to be ok to use the snp library"
> help(makeSnp) #access help file
> example(makeSnp) #see examples
```

A.2 Handling input

The `snpLayout` object must give sufficient informations about the experiment.

- Number of SNP
- Number of duplicates
- Space between duplicates
- Character: either "diso" or "triso"
- Files of arrays
- Names to be given to arrays (optional)
- Comments of the experimentation (optional)

If `type = "diso"`, people are supposed to have two chromosomes for the considered SNPs, if "triso", they are supposed to have three chromosome for the considered SNPs. **saveLayout** writes automatically an `.diso` extension in a coma separated value text. You can edit it using any text editor. **readLayout** loads a layout created by `saveLayout`. **addArray** adds new arrays to a layout.

In the following example and for the rest of this chapter we use the data set stored in the data folder of the package. This should allow you to redo this example.

```
> # finding our path to our data. for example
> array = c(
+ "/usr/local/lib/R/site-library/snp/data/SLIDE1.csv",
+ "/usr/local/lib/R/site-library/snp/data/SLIDE2.csv",
+ "/usr/local/lib/R/site-library/snp/data/SLIDE3.csv",
+ "/usr/local/lib/R/site-library/snp/data/SLIDE4.csv")
```

```

>
> # warnings : check default values
> # Check options (number of snp...)
> monLayout <- createLayout(arrays =array, +
+ comment = "example for the snp package",type="triso")
> monLayout
Layout for mic people modified the
Tue May 17 15:55:15 2005

      General informations:
nSnp          54
nDup          2
nSpace        1
nArrays       4

                        Arrays:
Array1 /usr/local/lib/R/site-library/snp/data/SLIDE1.csv
Array2 /usr/local/lib/R/site-library/snp/data/SLIDE2.csv
Array3 /usr/local/lib/R/site-library/snp/data/SLIDE3.csv
Array4 /usr/local/lib/R/site-library/snp/data/SLIDE4.csv

Comments: example for the snp package
> # uncomment to add an array
> # monLayout <- addArray(monLayout,"array to be added")
> # uncomment to save
> # saveLayout(monLayout,name = "monLayout" , "/path/")
> # use readLayout to load a previous layout
>

```

Files containing the data are read when creating the `snpRaw` object thanks to the **layout2Raw**. You can modify the default options and tell in which columns ID, green channel, red channel are stored. It is also possible to read any desired format by adding further arguments to the function (See help file.)

```

> # read input
> # take care of your format
> monSnp <- layout2Raw(monLayout,+
+ dec=",",sep="\t",green=6,red=7)
/usr/local/lib/R/site-library/snp/data/SLIDE1.csv
/usr/local/lib/R/site-library/snp/data/SLIDE2.csv

```

```
/usr/local/lib/R/site-library/snp/data/SLIDE3.csv
/usr/local/lib/R/site-library/snp/data/SLIDE4.csv
```

A.3 Normalization of the data

A.3.1 Problem

```
#####
# Part 1,5 : pb : on each file are stored several arrays
#####

# well..... that is not very a good news...
# please make sure you cannot try to avoid this confusion.
# t may be a source of a lot of errors
# anyway in our example that is what is happening and this
# lead to a code very confusing (my apologizes)
# here is a example of solution

# please next time use a _simpl_ id
id <- as.character(monSnp$data$id)
manip <- sub("ch12","",id)
manip <- sub("ch5","",manip)
manip <- sub("ch10","",manip)
manip <- sub(" "," ",as.character(manip))
manip <- sub(" "," ",as.character(manip))
manip <- strsplit(as.character(manip)," ")
nom <- NULL
for ( i in 1:length(id)) nom <- c(nom,manip[[i]][2])
manip <- sub(" ","",as.character(nom))
manip <- sub(" ","",as.character(manip))
manip <- as.factor(manip)
monSnp$data$array <- as.factor(manip)
```

A.3.2 Within and between normalizations

Use the **normSnp** function to normalize data. Possible value for within normalization are "mean", "loess" or "quantile". For between normalization, possible values are "green", "red", c("red","green") , "none". You may also produce plot to control the result of your normalization setting plot=TRUE.

We recommend the "quantile" normalization to classify people afterwards, experience shows that this within normalization leads to more distinct classes.

```
> monSnp <- normSnp(monSnp)
Within normalization of Array1
Within normalization of Array2
Within normalization of Array3
Within normalization of Array4
Between normalization for green channel
Between normalization for red channel
> # check default value and option in the help file
> # top produce plot , perhaps will not work on window
> # monSnp <- normSnp(monSnp,plot=TRUE)
> monSnp
Layout for triso mic people modified Tue May 17 15:56:14 2005
```

Current state:

```
Within normalization using quantile method
Between normalization for red channel
Between normalization for green channel
```

```
nId: 53 ; nArrays: 4 ; nObs: 5724
nSnp: 54 ; nDup: 2 ; nSpace: 1
```

Extract:

	id	type	red	green	array	nb
541	10020.1	(27/09/04)	D 359951.48	1886033.907	Array1	1
542	10020.1	(27/09/04)	D 320756.71	1752148.117	Array1	1
543	10020.1	(27/09/04)	D 97230.90	96712.871	Array1	1
544	10020.1	(27/09/04)	D 26667.87	88247.811	Array1	1
545	10020.1	(27/09/04)	D 38714.34	5250.714	Array1	1
... and 5719 more rows						

WARNING: observations have been normalized

The type of plot produced by the normSnp function are shown figures [2.2](#) and [2.3](#) page [17](#)

A.4 Classify SNP

A.4.1 Select a type

First select a type of people you want to analyze and use `raw2Res` function to create a `snpRes` object where your result will be stored

```
> trisoRaw <- selType(monSnp,"triso")
> mesRes <- raw2Res(trisoRaw)
> mesRes
Results for triso people modified Tue May 17 15:56:14 2005
nId: 21 ; nSnp: 54
0 Snp are already done
Last Snp done: 0
```

You can access to particular snp using the `print` function. By default, this function is hidden and print only the last done snp:

```
> mesRes
Results for diso people modified Fri May 6 11:39
nId: 22 ; nSnp: 54
54 Snp are already done
Last Snp done: 54
Details for Snp 54
Parameters:
  pro      mu      sd
1  1 0.4918416 0.00876837
      1      2      3      4      5      6      7
"yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
      9     10     11     12     13     14     15
"yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
     17     18     19     20     21     22
"yellow" "yellow" "yellow" "yellow" "yellow" "yellow"

> print(mesRes,snp=1)
Results for diso people modified Fri May 6 11:39
nId: 22 ; nSnp: 54
54 Snp are already done
Last Snp done: 54
Details for Snp 1
Parameters:
      pro      mu      sd
```

```

3 0.1136053 0.4707042 0.005454244
1 0.4204289 0.5080045 0.003649593
2 0.4659657 0.5297052 0.012985288
      1      2      3      4      5      6      7
"yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
      9     10     11     12     13     14     15
  "gray" "yellow" "yellow" "yellow" "yellow" "yellow"
      17     18     19     20     21     22
"yellow" "yellow" "yellow" "yellow" "yellow" "yellow"

```

A `writeSnp` function should allow you to write the whole classification in a standard format text file.

A.4.2 The `makeSnp` function

With `normSnp`, this is a core function where most of the statistical computation is done. Necessary arguments are the `snpRes` and `snpRaw` object and which snp is to be studied. You may also use the `flag` option to give a minimal values under which observations are flagged. If `'flag'` is `NULL`, a default value is used (See `optionSnp`). You may also give the number of classes, eg `'2:4'` for at least 2 classes and at most 4 classes. Or `'3'` for 3 classes exactly. If you give more than one possible value, the BIC criterium is used to determine automatically the number of cluster. Set `plot= TRUE` to produce plot and `force = TRUE` if the snp has already been studied. (Default to `FALSE` to prevent overwriting existing results.)

```

> # simplest way to use makeSnp
> # mesRes <- makeSnp(trisoRaw, mesRes , snp = 7)
> # define a subset of "good quality" observation
> # mesRes <- makeSnp(trisoRaw, mesRes , +
+ snp = 7 , flag2D = 0.3)
> # try all possible model for at most 4 class
> # and choose them using ILC criterium
> tmp = makeSnp(trisoRaw, mesRes , snp = 4 , +
+ plot= TRUE , model = "EIE" , flag2D = 1)
ILC for EIE with 1 classes equals to -186.4631
ILC for EIE with 2 classes equals to -1564.644
ILC for EIE with 3 classes equals to -5101.637
ILC for EIE with 4 classes equals to -7631.223

```

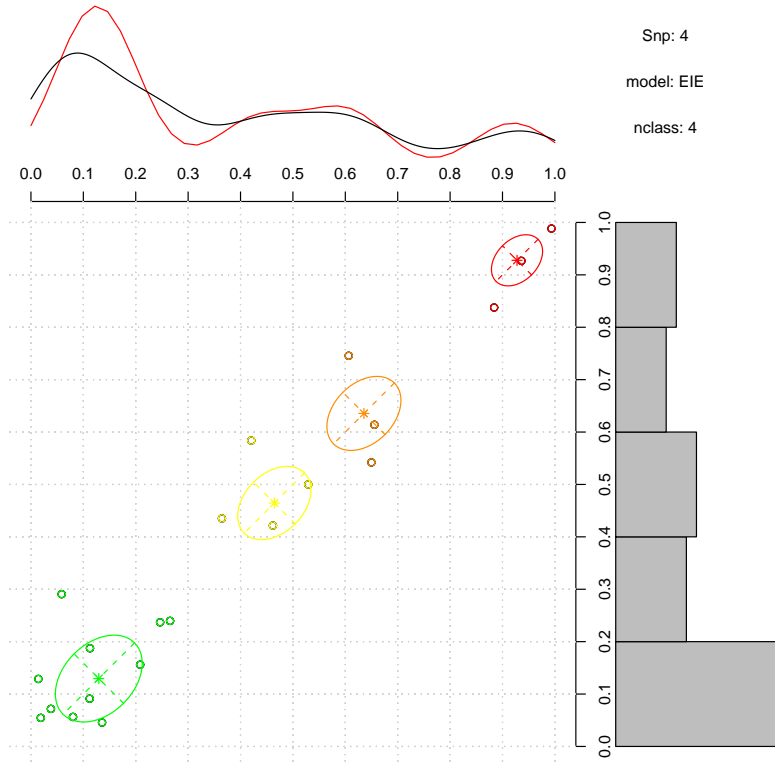


Figure A.1: Type of plot produce by the `makeSnp` function. Value that are surrounded by a gray circle have not been used in the computation of the algorithm (`flag1D` and `flag2D` option). On top, two density estimation using kernel density (See page 44 for further details) and on the right a histogram

Best model EIE with 4 classes

Warning messages:

1: Singular matrix in sep 3

stopping algorithm in: `emSnp(data, subset = subset, model = model_i, nclass = i, plot = FALSE)`

2: Singular matrix in sep 1

stopping algorithm in: `emSnp(data, subset = subset, model = model_i, nclass = i, plot = FALSE)`

Note that the R console warns you when the algorithm is degenerating. The criterion used to see whether the algorithm is leading to singular covariance matrix is given in the option `Snp(eps)`.

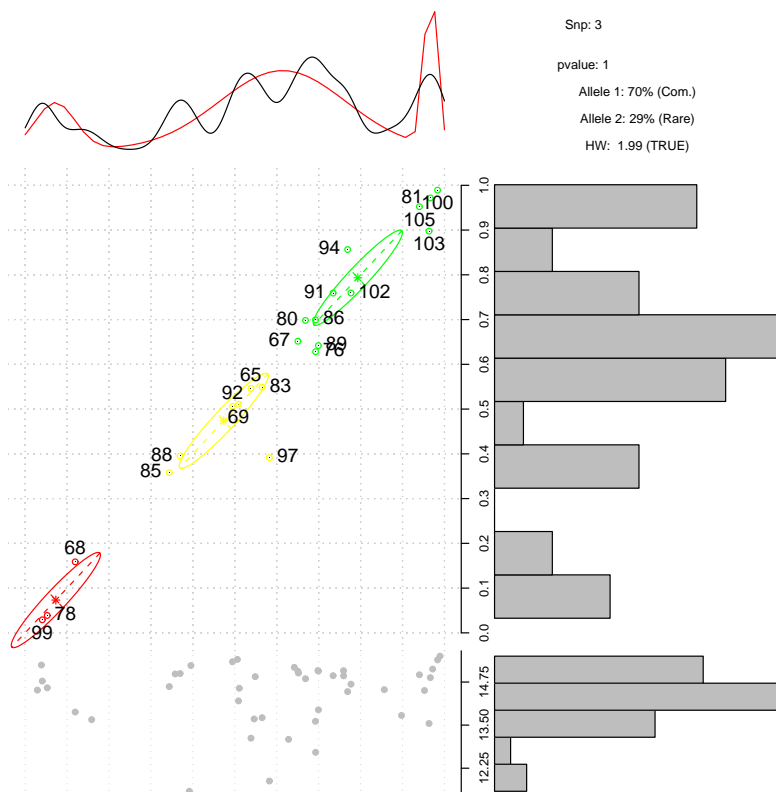


Figure A.2: Example of plot produce by interfaceSNP: you can modify the classification and attribute to particular observation a appropriate class. Those data will get a triangle instead of a circle and their pvalue will be removed. You can also get the name of certain ID. In the right upper corner, some informations about alleles are given (See page 44)

A.4.3 Access classification

You may want to modify the classification or to retrieve some informations about your results. Use the *interfaceSnp* function (See figure A.2 page 57).

```
> interfaceSnp(trisoRaw,mesRes, snp = 1 )
Results for triso people modified Wed Jul 6 15:53:39 2005
nId: 21 ; nSnp: 54
1 Snp are already done
Last Snp done: 1
Details for Snp 1
Parameters:
NULL
```

```

10002.0 (21/10/04)          10003.0 (21/10/04)
      "red"                  "yellow"
10004.0 (11/10/04)          10005.0 (04/10/04)
      "yellow"               "yellow"
10006.0 (21/10/04) 10006.0 (21/10/04) taq eppendorf
      "yellow"               "yellow"
10009.0 (15/11/04)          10009.0 (15/11/04) qte0,5
      "yellow"               "yellow"
10010.0 (21/10/04)          10011.0 (15/11/04)
      "yellow"               "yellow"
      10012.0 15/11/04)      10012.0 (15/11/04) qte red
      "red"                  "red"
10013.0 (15/11/04)          10013.0 (15/11/04) qte red
      "red"                  "red"
10014.0 (21/10/04)          10016.0 (11/10/04)
      "darkorange"           "darkorange"
10017.0 (21/10/04)          10019.0 (15/11/04)
      "yellow"               "yellow"
10019.0 (15/11/04) qte0,5  10020.0 (11/10/04)
      "red"                  "darkorange"
10021.0 (04/10/04)
      "yellow"

```

```

> # checking pvalue
> # be careful with pvalue (if you set a pvalue too low, you
> # cannot go back there use " <- " when you are sure)
> mesRes <- interfaceSnp(trisoRaw,mesRes, snp = 1, pvalue = 0.05 )
>
>
> # if you want to assign certain value to the darkorange class (AAB)
> # left click on value (right click to exit)
> mesRes <- interfaceSnp(trisoRaw,mesRes, snp = 1, pvalue = 0.05 , identify
> # possible vlaue are "green" "gray" "red" "yellow" and for trisomic "darkor
> # those data will be marked with a triangle
>
> # get the ID
> interfaceSnp(trisoRaw,mesRes, snp = 1, pvalue = 0.05 , identify = "ID")
Results for triso people modified Wed Jul 6 15:53:39 2005
nId: 21 ; nSnp: 54
1 Snp are already done
Last Snp done: 1

```

Details for Snp 1

Parameters:

NULL

10002.0 (21/10/04)	10003.0 (21/10/04)
"red"	"yellow"
10004.0 (11/10/04)	10005.0 (04/10/04)
"yellow"	"yellow"
10006.0 (21/10/04)	10006.0 (21/10/04) taq eppendorf
"yellow"	"yellow"
10009.0 (15/11/04)	10009.0 (15/11/04) qte0,5
"yellow"	"yellow"
10010.0 (21/10/04)	10011.0 (15/11/04)
"yellow"	"yellow"
10012.0 15/11/04)	10012.0 (15/11/04) qte red
"red"	"red"
10013.0 (15/11/04)	10013.0 (15/11/04) qte red
"red"	"red"
10014.0 (21/10/04)	10016.0 (11/10/04)
"darkorange"	"darkorange"
10017.0 (21/10/04)	10019.0 (15/11/04)
"yellow"	"yellow"
10019.0 (15/11/04) qte0,5	10020.0 (11/10/04)
"red"	"darkorange"
10021.0 (04/10/04)	
"yellow"	

>

> # export result

> writeRes(mesRes)

A.5 Options

A.5.1 Update

To limit lost of time, when new observations are available, you can redo your classification in one update command, provided that the previous classification had given relevant results. Here is an example of update with other observations. These observations can be found in the folder "pour adrien" at following adress: <ftp://ftp.igh.cnrs.fr/users/Emmanuelle/>.

```

# creation layout ok
Layout <- createLayout(56,2,1,
arrays=c("array/lame3.csv"), name = "array3")
Layout <- addArray(Layout,
arrays=c("array/lame5.csv","array/lame6.csv",
"array/lame7.csv","array/lame8.csv"),
names =c("array5","array6","array7","array8"))

# omitting lame4M
LameForm <- layout2Raw(Layout, id=2, green=6,
red=7, type=1, header=TRUE, sep=";", dec=",")
TypeForm <- selType(LameForm,"diso")
resForm <- raw2Res(TypeForm)
for (i in 1:56)
resForm <- makeSnp(TypeForm,resForm, i,
plot=TRUE,force=TRUE)
# on sauve et on quite
saveLayout(Layout)
infoSnp(resForm,"info")

```

Note the infoSnp function which give all the informations about the algorithm applied to SNPs. When no name of file are given, informations are just displayed in the R console. Otherwise, when a name of file is given, informations are written in a file. This file must be afterwards reloaded to update the classification when new observations are available:

```

library(snp)
Layout <- readLayout("layout.snp")
Layout <- addArray(Layout,
arrays=c("array/lame4.csv"),names =c("array4"))

# new array added
LameForm <- layout2Raw(Layout, id=2, green=6,
red=7, type=1, header=TRUE, sep=";", dec=",")
LameForm <- normSnp(LameForm) #normalisation
TypeForm <- selType(LameForm,"diso") # selction
res <- updateSnp(TypeForm , read.table("info"))
writeRes(res) # save result in cvs format

```

Other options are available:

- snpRaw: An object of class snpRaw where data are stored

- info: An object where are stored informations about SNPs, see infoSnp
- snp: Integer vector giving the snps that are to be updated. By default all snp are updated.
- plot: Logical
- remove: If TRUE, if in a previous step you have flagged values, same flags will be applied

A.5.2 General options

You can access to general option through the **optionSnp** function. You can specify :

- the default boundaries of the diso classes used to determine thanks to the mean of a class its color
- the default boundaries of the triso classes used to determine thanks to the mean of a class its color.
- the default scalar tolerance unused in the EM algorithm
- the default tolerance used in the EM algorithm
- the default maximal number of iterations for the EM algorithm
- the default names of the model for the EM algorithm. Either "E" or "EEE" or "EIE"
- the default method to normalize the data. Either "quantile" or "mean"

Appendix B

Help files

Package ‘snp’

July 12, 2005

Title find SNP for disomic people

Version 2.1.1

Date 2005-13-04

Author Adrien du Moulinet d’Hardemare

Description Normalize data and implement an EM algorithm to classify SNPs

Maintainer adrien d’hardemare <adrien.dhardemare@gmail.com>

Depends R (>= 2.0), mclust, splines

License GPL

R topics documented:

emSnp	2
estepEEEsnp	3
infoSnp	3
interfaceSnp	4
layout2Raw	5
makeSnp	6
mstepEEEsnp	7
normBetween	8
normSnp	9
normalizeLimma	11
optionSnp	12
createLayout	13
selType	14
disoLayout	15
snpRaw	16
snpRes	17
snpSnp	18
updateSnp	19
writeRes	20

Index	21
--------------	-----------

Examples

```
# not called by user
```

```
estepEEEsnp      intern function
```

Description

fixing a small bug. Cf [estepEEE](#). E step in the EM algorithm.

```
infoSnp          Give informations on SNP
```

Description

Tell which SNP have been done

Usage

```
infoSnp(snpRes , file = NULL , ...)
```

Arguments

<code>snpRes</code>	An object of class <code>snpRes</code>
<code>file</code>	If file is not NULL, result will be written in a standard format in file 'file'
<code>...</code>	Further arguments for the <code>write.table</code> function

Value

Return a `data.frame` where informations on SNP are stored

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programmation applied to Microarrays, 2005, Masterthesis

See Also

`write.table`

Examples

```

data(disoRaw)
disoRes <- raw2Res(disoRaw)                                # where to store result
disoRes = makeSnp(disoRaw, disoRes, 1)                    # modelization
disoRes = interfaceSnp(disoRaw, disoRes, 1)               # interpretation
infoSnp(disoRes)                                         # only the first element is d

```

interfaceSnp *Provide a simple interface to manipulate data directly*

Description

Allow a visualization of the data and of the classification and to access to classification to correct it.

Usage

```
interfaceSnp(snpRaw, snpRes, snp, identify = TRUE, pvalue = NULL, flag1D=0, flag2D =
```

Arguments

snpRaw	data of class snpRaw
snpRes	classification of class snpRes
snp	number of the snp(s)
identify	character string : allow access to classification. See details
pvalue	control the uncertainty of the classification
flag1D	data for wich $\log(\text{green} + \text{red}) < \text{flag1D}$ are not used in to update model
flag2D	data which correlation is superior to $\text{flag2D} * \text{mean correlation}$ are not used in to update model

Details

identify is a string naming a particular class (eg: "gray", "red", "green", "yellow" and for trisomic "darkorange") or "ID" to get the ID of the data. If you enter the name of a class, click with the left button of the mouse on the graph on the points you want to classify as "identify". Click on the right button of the mouse to exit the identify mode. Note that if you classify data using identify mode, uncertainty probability for those data will be removed.

pvalue allow you to choose a maximal uncertainty probability over which data will be classify as "gray". Please note that for the instant, you cannot return to a previous classification using a previous pvalue if the previous was upper. Uncertainty probability are calculated in the makeSnp step and adjusted to the design.

Value

an object of class snpRes

Note

check the pvalue

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[makeSnp](#)

Examples

```
data(trisoRaw)
trisoRes <- raw2Res(trisoRaw)
trisoRes <- makeSnp(trisoRaw, trisoRes, 2)
interfaceSnp(trisoRaw, trisoRes, 2, flag1D=10)
```

layout2Raw

layout2Raw

Description

Use informations in a layout object to find data, read them and store them

Usage

```
layout2Raw(layout, type = 1, id = 3, green = 2, red = 3, header = TRUE, ...)
```

Arguments

layout	name of the layout. Of class layoutSnp
id	integer: column where ID are stored
green	integer: column where green light are stored
red	integer: column where red light are stored
header	do the data files contain headers
...	further arguments

Details

...are passed to read.table function and may be of some use if the format is not standard.

Value

returns a `snpRaw` object

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

`snpRaw`, `read.table`

Examples

```
example(saveLayout)
layout2Raw(layout)
```

makeSnp

Calcul the model used to classify data

Description

Interface to the `emSnp` function.

Usage

```
makeSnp(snpRaw, snpRes, snp, flag1D = 0, flag2D = Inf, nclass = NULL, plot = FALSE,
```

Arguments

<code>snpRaw</code>	an object of class <code>snpRaw</code> where data are stored
<code>snpRes</code>	an object of class <code>snpRes</code> where results are stored
<code>snp</code>	which <code>snp</code> is to be modeled
<code>flag1D</code>	data for which $\log(\text{green} + \text{red}) < \text{flag1D}$ are not used to find the classes
<code>flag2D</code>	data which correlation is superior to $\text{flag2D} * \text{mean}$ are not used to find classes
<code>nclass</code>	a numeric vector giving the number of possible classes
<code>plot</code>	logical
<code>force</code>	logical : if 'snp' has already been modeled, should we overwrite results ?
<code>model</code>	character string. See details
<code>crit</code>	character string. See details
<code>verbose</code>	Mostly for intern convenience
<code>...</code>	Not used

Details

Classes are found using a model of mixture of gaussian for observations replicated twice: Name of the model is either :

"E" for spherical, equal variance (one-dimensional)

"EEE": ellipsoidal, equal volume, shape, and orientation

"EIE": ellipsoidal, varying volume, shape and orientation equal "E" and "EEE" are more appropriate if the quality of the data is low. Otherwise, use "EIE". To choose between the various models and the optimal number of class, you can specify a criterion. Either "ILC" for integrated likelihood criterion or "BIC" for bayesian information criterion.

Value

return a snpRes object

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[mstepEIEsnp](#), [mstepEEEsnp](#)

Examples

```
data(trisoRaw)
trisoRes <- raw2Res(trisoRaw)
makeSnp(trisoRaw, trisoRes, 2, plot=TRUE , flag1D = 10, force=TRUE)
```

mstepEEEsnp

mstep for SNP

Description

Implement the M step adapted to the analysis of SNP data. EEE and EIE model.

Usage

```
mstepEEEsnp(data, z, ...)
```

Arguments

data	matrix with two columns. In row observations, in columns, duplicates
z	matrix of conditionnal probability. The number of columns is equal to the number of classes
...	Not used

Value

A list containing the mean, sigma and proportions of each class.

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programmation applied to Microarrays, 2005, Masterthesis

See Also

for the E model [mstepE](#)

Examples

```
# not called by user
```

normBetween	<i>Normalize between array the observations</i>
-------------	---

Description

Produce a quantile normalization method to normalize the SNP

Usage

```
normBetween(data, factor, plot = FALSE , spar = 0.6)
```

Arguments

data	A numeric vector. NA allowed
factor	A factor vector giving the arrays of the data
plot	Logical

Details

Array may have a different number of observations. Finding the value of the quantile is done through a spline approximation.

Value

return the normalized data

Note

peut etre conflit avec normalization inter, parameter pour smooth ?, est ce que cela donne des bons resultats ?

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[normSnp](#), [smooth.spline](#)

Examples

```
# trying the method for between normalization
data(snpRaw)

temp <- normBetween(snpRaw$data$red, snpRaw$data$array, plot=TRUE)
temp <- normBetween(snpRaw$data$red, snpRaw$data$array, plot=TRUE, spar=0.9)
```

normSnp

Normalization of data

Description

Will permit afterwards to make a relevant comparison between red and green light

Usage

```
normSnp(x, force=FALSE, tie=FALSE, within="quantile", between=c("none"), ...)
```

Arguments

<code>x</code>	Object of class <code>snpRaw</code> where data are stored
<code>plot</code>	Logical: should a graph for each array be produced.
<code>tie</code>	Logical: intern argument passed into <code>normalizeLimma</code>
<code>Within</code>	Either "mean" or "quantile" or "loess". Used in within normalization
<code>Between</code>	Either "red" or "green" or both or "none". Used in between normalization
<code>...</code>	Further arguments to plot the graphics

Details

Data are normalized according to the arrays they belong to (ie according to the date they were made). If "mean", a mere translation is produced so as to ensure that the green mean and the red mean are equal. If "quantile", a quantile normalization is produced through `normalizeLimma`. `normSnp` ensures that data have not been already normalized. Use the `force` option, if they were already normalized.

Value

Return an object of class `snpRaw`

Note

seems ok with between but should perhaps implement a loess function

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[snpLayout](#) [normalizeLimma](#), [optionSnp](#)

Examples

```
data(disoRaw)
temp <- normSnp(disoRaw, within= "loess" , plot=TRUE, between = c("red", "green") )
temp <- normSnp(disoRaw, within= "loess" , plot=TRUE, between = "none")
```

normalizeLimma	<i>Quantile normalization</i>
----------------	-------------------------------

Description

Ensure that two distributions have the same quantiles. Called by `normDiso`. The user does not need to access to it.

Usage

```
normalizeLimma(A, ties = TRUE)
```

Arguments

A	A matrix with two columns
ties	Logical

Details

The two columns of A are normalized using a quantile normalization process. If 'tie', a special treatment is reserved to the extrem values.

Value

returns a matrix

Author(s)

Gordon Smyth

References

Gordon K. Smyth, 2003, Linear models and empirical bayesian methods for assessing differential expression in microarray experiments.

B. M. Bolstad ,R. A. Irizarry, M. Astrand and T. P. Speed , 2002 , A comparison of normalization methods for high density oligonucleotide array data based on variance and bias,

See Also

[normalizeBetweenArrays](#) (package 'limma'), [normDiso](#)

`optionSnp`*General options*

Description

Access to options and change them

Usage

```
optionSnp(...)
```

Arguments

<code>disoClass</code>	A vector of type <code>c(a,b)</code> giving the boundaries of the classes. If a value is between 0 and a (resp a and b or b and 1), it belongs to the first (resp second or third) class
<code>trisoClass</code>	A vector of length of type <code>c(a,b,c)</code> giving the boundaries of the classes. If a value is between 0 and a (resp a and b or b and c or c and 1), it belongs to the first (resp second or third or fourth) class
<code>eps</code>	For EIE model. Stop algorithm if volume of the biggest class is superior to <code>eps * volume of the smallest class</code> . Prevent degenerating algorithm.
<code>tol</code>	Tolerance used in the EM algorithm (of <code>length2</code>)
<code>itmax</code>	Maximal number of iterations for the EM algorithm
<code>emModelNames</code>	Names of the model for the EM algorithm. Either "E" or "EIE" or "EEE"
<code>norm</code>	Default method to normalize the data. Either "quantile" or "mean"
<code>spar</code>	Penalty value used for spline in between normalization

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[EMclust](#), [p.adjust](#), [smooth.spline](#)

Examples

```
## setting default values
optionSnp (
disoClass = c(0.33,0.66) ,
trisoClass= c(0.25,0.50,0.75),
pvalue= 0.05,
eps= 5,
tol= c(5e-8,5e-8),
itmax= c(1000,1000)
emModelNames= c("EEE", "EIE"),
norm= "quantile",
spar = 0.6
)
```

createLayout *Informations about the design*

Description

Display important informations concerning the design of the experimentation

Usage

```
createLayout(nSnp = 54, nDup = 2, nSpace = 1, arrays, names = NULL, comments = NULL)
addArray(layout, arrays, names = NULL)
saveLayout(layout, name="layout", path="")
readLayout(file, sep=" ", ...)
```

Arguments

nSnp	Number of SNPs
nDup	Number of duplicates
nSpace	Space between duplicates
arrays	Files of arrays
names	Names to be given to arrays (optional)
comments	Comments of the experimentation (optional)
date	If NULL the current date is supposed
layout	An object of class snpLayout
name	Name of the saved layout
path	Path where the layout will be saved
file	Complete name of the file to be read
sep	The field separator character

Details

createLayout creates a layout object. If type = "diso", people are supposed to have two chromosomes for the considered SNPs, if "triso", they are supposed to have three chromosomes for the considered SNPs. saveLayout writes automatically an .snp extension in a comma separated value text. You can edit it using any text editor. readLayout loads a layout created by saveLayout addArray adds new arrays to a layout

Value

Returns a snpLayout object

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[snpLayout](#) [write](#), [write.table](#), [scan](#), [read.table](#)

Examples

```
#Layout <- createLayout(54,2,1,c("/path/to/array1", "/path/to/array2"),names=c('array1', 'array2'))
#saveLayout(Layout,path=~/" )
#rm(Layout)
#Layout <- readLayout("~/layout.snp")
#Layout <- addArray(Layout,c("one more array", "two more array"))
#Layout
```

selType

Select a type of data

Description

Select disomic or trisomic people among a whole data set

Usage

```
selType(snpRaw, type)
```

Arguments

snpRaw an object of class snpRaw
type character string. Either "diso" or "triso"

Value

return a object of class snpRaw

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

Examples

```
# cf demo.R
```

disoLayout *class disoLayout*

Usage

```
disoLayout(layout, ...)
```

Arguments

layout object of class disoLayout
... further arguments to be passed

Details

disoLayout stores informations concerning the layout of an experiment print.disoLayout prints it in a suitable way

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also[createLayout](#)**Examples**

```
Layout <- createLayout(3,2,1,"/hello","bonjour","hallo")
Layout
Layout$general
Layout$arrays
Layout$comments
```

*snpRaw**snpRaw*

Description

Class to store the raw observations. `writeSnpRaw` writes data in a file.

Usage

```
print.snpRaw(x, n = 5, ...)
writeSnpRaw(x, file, ...)
```

Arguments

<code>x</code>	object of class <code>snpRaw</code> to be printed
<code>n</code>	how many rows should be printed
<code>file</code>	File where to write data
<code>...</code>	further arguments

Details

You may access to values through `x[i,j]` with `i` the SNP you are interested in and `j` wether "green", "red", "ratio", "log" to access to the green, red, $\text{green}/(\text{red}+\text{green})$, $\log(\text{red}+\text{green})$ values.

Value

An object of class `snpRaw`

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programmation applied to Microarrays, 2005, Masterthesis

See Also

[layout2Raw](#), [write](#)

Examples

```
data(snpRaw)
snpRaw
print(snpRaw, 10)
plot(snpRaw[3, "ratio"], snpRaw[3, "log"], main="SNP 3")
```

snpRes

SnpRes

Description

Convenient class to store the classification on SNP

Usage

```
raw2Res(snpRaw)
print(snpRes, snp=snpRes$lastSnp, id=1:snpRes$nId)
writeSnpRes(snpRes, file, ...)
```

Arguments

snpRaw	an object of class <code>snpRaw</code> to store data
snpRes	an object of class <code>snpRes</code> to store results
snp	snp to be printed
id	id to be printed
file	name of the file

Details

`snpRes` contains the result of the classification. *snpRes* class contains the class of the people, *snpRes* proba the corresponding proba and *snpRes* `snp[[i]]` information about the snp *i*

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

Examples

```
data(snpRaw)
snpRaw <- normSnp(snpRaw)
snpRes <- raw2Res(snpRaw)
snpRes
print(snpRes, 1:3, 2:4)
```

snpSnp	<i>Class to access data</i>
--------	-----------------------------

Usage

```
snpSnp(snpSnp, nDup = c(1:2), nSnp = c(1), nId = c(1:2), ...)
```

Arguments

lame	An object of class snpSnp
nDup	Vector of duplicates
nSnp	Vector of SNPs
nId	Vector of id
...	further arguments

Details

snpSnp is mainly a list of two arrays, 'red' and 'green' where data are stored in three dimensions: according to their id, their snp and their duplicates.

Note

to be tested

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[raw2Snp](#)

Examples

```
data(snpRaw)
snpRaw <- normSnp(snpRaw)
snp <- raw2Snp(snpRaw)
snp
print(snp, c(10, 13), c(5:7), c(1:2))
```

updateSnp *Update a classification*

Description

Assuming that number of classes and name of model have been found in a previous analysis, this function update a classification after adding new arrays and a new normalizations.

Usage

```
updateSnp(snpRaw, info, snp=1:snpRes$nSnp, plot = FALSE, remove = FALSE)
```

Arguments

snpRaw	An object of class snpRaw where data are stored
info	An object where are stored informations about SNPs, see infoSnp
snp	Integer vector giving the snps that are to be updated. By default all snps are updated.
plot	Logical
remove	If TRUE, if in a previous step you have flagged values, same flags will be applied

Value

Return an object of class snpRes

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[makeSnp](#), [infoSnp](#)

Examples

writeRes

Write classification in a table

Description

Write results of a classification in a standard csv format

Usage

```
writeRes(snpRes, file = "resuts.snp.csv" , ...)
```

Arguments

snpRes	An object of class snpRes where results of the classification is stored
file	Name of the file
...	Further arguments passed into write.table

Author(s)

Adrien d'Hardemare

References

A. d'Hardemare, Statistical Classification and Programming applied to Microarrays, 2005, Masterthesis

See Also

[write.table](#)

Examples

```
## The function is currently defined as
#function (snpRes , file ,...){
#write.table (snpRes$class,file=file, ...)
# }
```

Index

- *Topic **kwd1**
 - mstepEEEsnp, 7
 - selType, 14
- *Topic **kwd2**
 - mstepEEEsnp, 7
 - selType, 14
- *Topic **addArray**
 - createLayout, 13
- *Topic **createLayout**
 - createLayout, 13
- *Topic **datasets**
 - snpRes, 17
- *Topic **kwd1**
 - emSnp, 1
 - infoSnp, 3
 - interfaceSnp, 4
 - makeSnp, 6
 - normBetween, 8
 - optionSnp, 12
 - updateSnp, 19
 - writeRes, 20
- *Topic **kwd2**
 - emSnp, 1
 - infoSnp, 3
 - interfaceSnp, 4
 - makeSnp, 6
 - normBetween, 8
 - optionSnp, 12
 - updateSnp, 19
 - writeRes, 20
- *Topic **layout2Raw**
 - layout2Raw, 5
- *Topic **print.snpRaw**
 - snpRaw, 16
- *Topic **readLayout**
 - createLayout, 13
- *Topic **saveLayout**
 - createLayout, 13
- *Topic **snpRaw**
 - normSnp, 9
 - snpRaw, 16
- addArray (*createLayout*), 13
- createLayout, 13, 16
- disoLayout, 15
- EMclust, 12
- emSnp, 1
- estepEEEE, 2
- estepEEEsnp, 2
- infoSnp, 3, 19
- interfaceSnp, 4
- layout2Raw, 5, 17
- makeSnp, 2, 5, 6, 19
- mstepE, 8
- mstepEEEsnp, 2, 7, 7
- mstepEIEsnp, 2, 7
- mstepEIEsnp (*mstepEEEsnp*), 7
- normalizeBetweenArrays, 11
- normalizeLimma, 10, 11
- normBetween, 8
- normDiso, 11
- normSnp, 9, 9
- optionSnp, 10, 12
- p.adjust, 12
- print.disoLayout (*disoLayout*), 15
- print.snpRaw (*snpRaw*), 16
- print.snpRes (*snpRes*), 17
- raw2Res (*snpRes*), 17
- raw2Snp, 18
- read.table, 6, 14
- readLayout (*createLayout*), 13

`saveLayout (createLayout)`, 13
`scan`, 14
`selType`, 14
`smooth.spline`, 9, 12
`snpLayout`, 10, 14
`snpRaw`, 6, 16
`snpRes`, 17
`snpSnp`, 18

`updateSnp`, 19

`write`, 14, 17
`write.table`, 14, 20
`writeRes`, 20
`writeSnpRaw (snpRaw)`, 16
`writeSnpRes (snpRes)`, 17