# Sample of aCGH analysis

ML Smith

November 28, 2005

## 1 Introduction

This document outlines some of the commands used to read in, investigate and subsequently segment array CGH data. The files analysed represent 50 breast cancer cell lines obtained from Jessica M Pole and Paul AW Edwards.

```
> options(width = 75)
> library(snapCGH)
```

*snapCGH* is designed to be used in conjunction with *limma* and so it will automatically load that library before proceeding.

## 2 Reading Data

We read in the samples and create the initial RG object using the following commands.

```
> datadir <- system.file("testdata", package = "snapCGH")
> targets <- readTargets("targets.txt", path = datadir)
> RG1 <- read.maimages(targets$FileName, path = datadir, source = "genepix")

Read c:/TEMP/Rinst.964/snapCGH/testdata/10Mbslide28.gpr
Read c:/TEMP/Rinst.964/snapCGH/testdata/10Mbslide29.gpr
```

When aCGH experiments are carried out, the reference channel is dyed using Cy5 and the test channel is dyed using Cy3.

Information about the clones on the array (e.g. what part of the genome they represent) can be read in separately using the `read.clonesinfo` function. In order to do this it is necessary to create a clones info file. Such a file (which can be created using Excel and saved as a 'txt' file) must contain columns called Position and Chr which give the position along a chromosome (in kb) and the chromosome to which a clone belongs. It must be ordered in the same way as the clones are ordered in the genepix file. The information is added to the `RG$genes` object. The second command adds information about the structure of the slide (blocks/rows/columns) to the `RG` object. Finally we read in a spot types file.

This file contains information about the control status of particular spots on the array and allows specific spots to be highlighted in many of the plotting functions. The content of a spot types file is covered extensively within the *limma* manual.

```
> RG1 <- read.clonesinfo("cloneinfo.txt", RG1, path = datadir)
> RG1$printer <- getLayout(RG1$genes)
> types <- readSpotTypes("SpotTypes.txt", path = datadir)
> RG1$genes$Status <- controlStatus(types, RG1)
```

```
Matching patterns for: ID Name Chr
Found 21 CTD-
Found 9 CTA-
Found 405 Chrom8
Found 66 Chrom3
Found 75 Chrom1
Setting attributes: values Color
```

We now proceed to use some information provided by control spots to remove clones which we deem to be of a poor quality. The clone spots which we will use as our control are drosophila clones. We remove clones whose intensity is too low relative to the control spots. (The background correction is carried out within this function.)

```
> RG2 <- backgroundCorrect(RG1, method = "minimum")
```

Next, we normalise the data. Here we will carry out a (global) median normalisation. Other options for normalization methods are: *none*, *loess*, *printtiploess*, *composite* and *robustspline*. The output of the normalization function is a new type of object called an *MAList*. This is composed of the $log_2$ ratios, intensities, gene and slide layout information which it gleans from the `RG` object.

```
> MA <- normalizeWithinArrays(RG2, method = "median")
```

We are now ready to process the data with the purpose of segmenting the dataset into regions corresponding to sections of the genome where there are the same number of copy number gains or losses.

Firstly, we use the `process.MAList` to 'tidy up' the MAList object. The *unmapScreen* option removes those clones which are incompletely mapped and the *dupRemove* option indicates whether clones with duplicate names should be averaged and removed from the dataset leaving only one occurence of each duplicated set (this option defaults to TRUE).

```
> MA <- process.MAList(MA)
```

```
Averaging duplicated clones
```

The next step is to overcome the problem that the segmentation functions (as written at present) can not cope with missing values. Hence, we use an interpolating method to overcome this problem. This is carried out using the `impute.lowess` function.

```
> MA2 <- impute.lowess(MA)
```

We are now ready to fit the segmentation method. For larger data sets this step can take a long time (several hours). We fit the HMM using the following commands. The other segmentation methods included in this library are *GLAD* and *DNAcopy*. See manual pages for the functions `run.DNAcopy` and `run.GLAD` for details of how to use them.

```
> SegInfo <- fitHMM(MA2, criteria = "AIC")
```

We now deal with the fact that the HMM sometimes has a tendency to fit states whose means are very close together. We overcome this problem by merging states whose means are within a given threshold. There are two different methods for carrying out the merging process. For more information on their differences please see the appropriate page in the helpfiles.

```
> SegInfo.merged <- mergeSegList(SegInfo, MergeType = 1)
```

After merging, the next step is to find any points of particular interest amongst the data. For instance outliers, focal aberrations and transitions. The function `find.genomic.events` performs this task and stores the information in an object of class `GEList`.

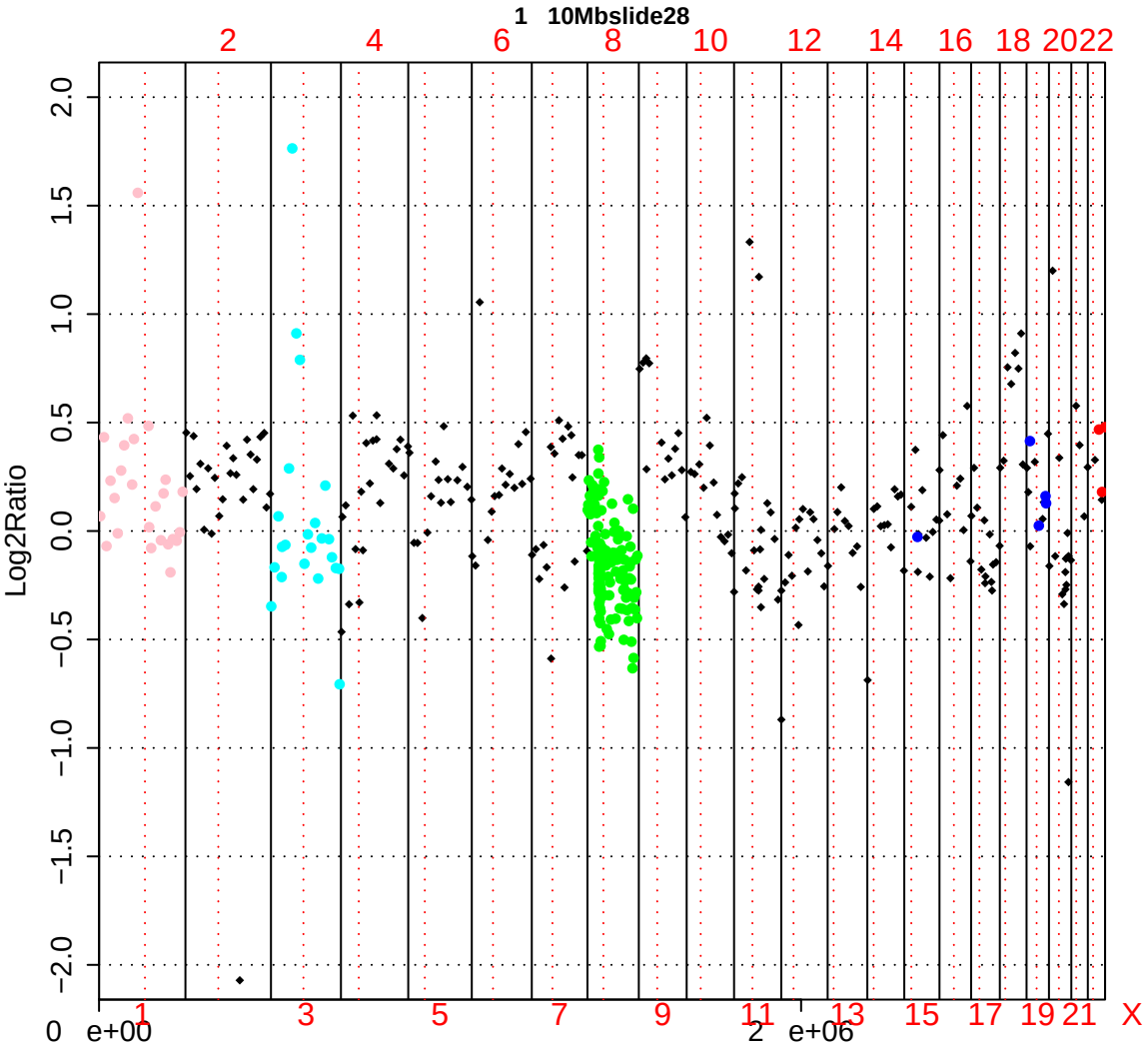
```
> GenList <- find.genomic.events(SegInfo.merged)
```

We are now ready to use any of the plotting functions available in the library.

# 3 Plotting Functions

The library comes with a variety of plotting functions that provide visual representations of the data at various stages of the analysis process. Firstly we will look at the `plotGenome` function. This function takes either an `MAList` or a `SegList` object (in this example we've used an MAList) and plots the M-value for each gene against it's position on the genome. The `array` argument indicates which array is plotted. This function utilizes the spot types data that was read in earlier to highlight specific genes of interest.

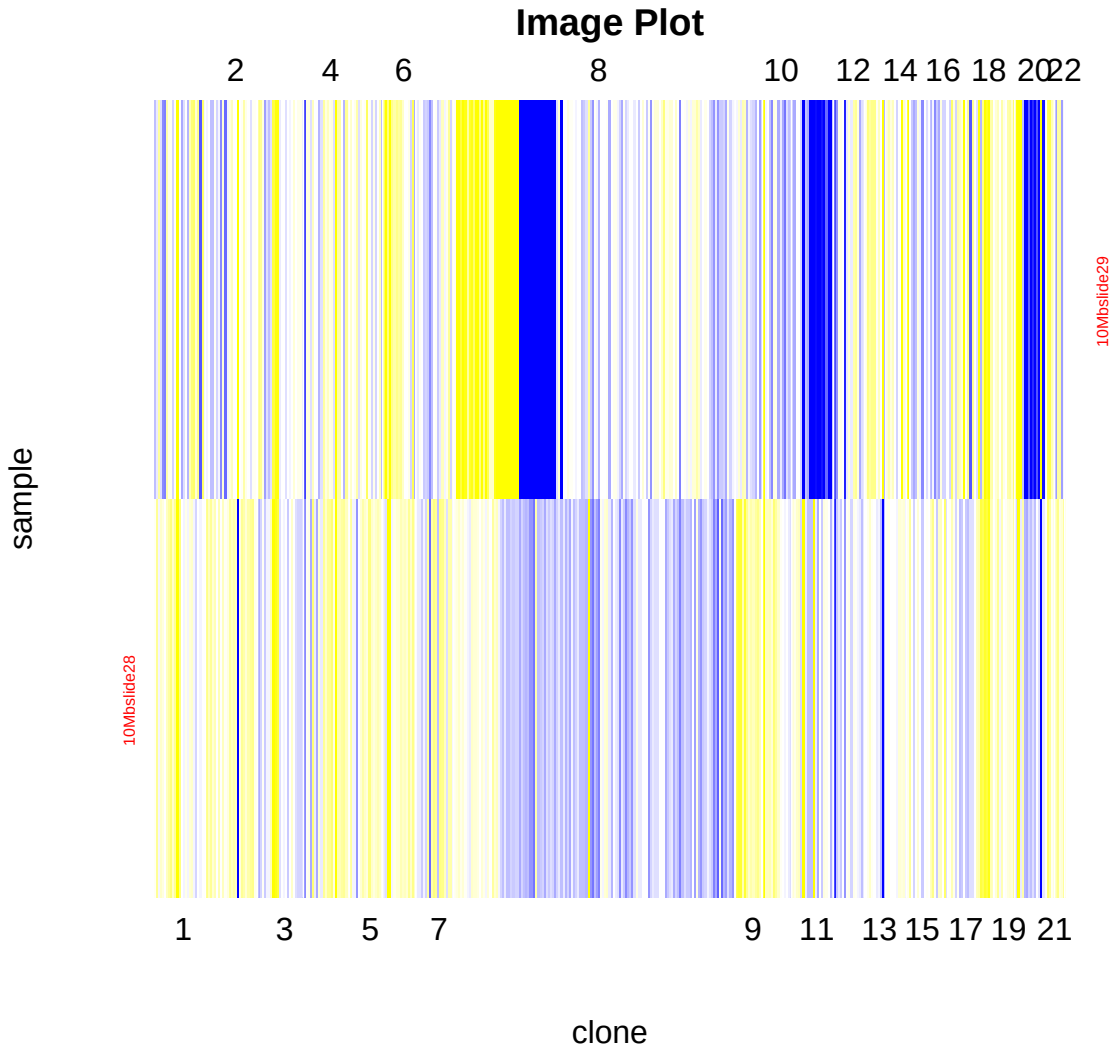
```
> plotGenome(MA2, array = 1)
```




Using the `labelled.genome.plot` function it is possible to look at specific chromosomes, rather than the entire genome as in the previous function. Which particular chromosome is to be plotted is specified using the `chrom.to.plot` argument.

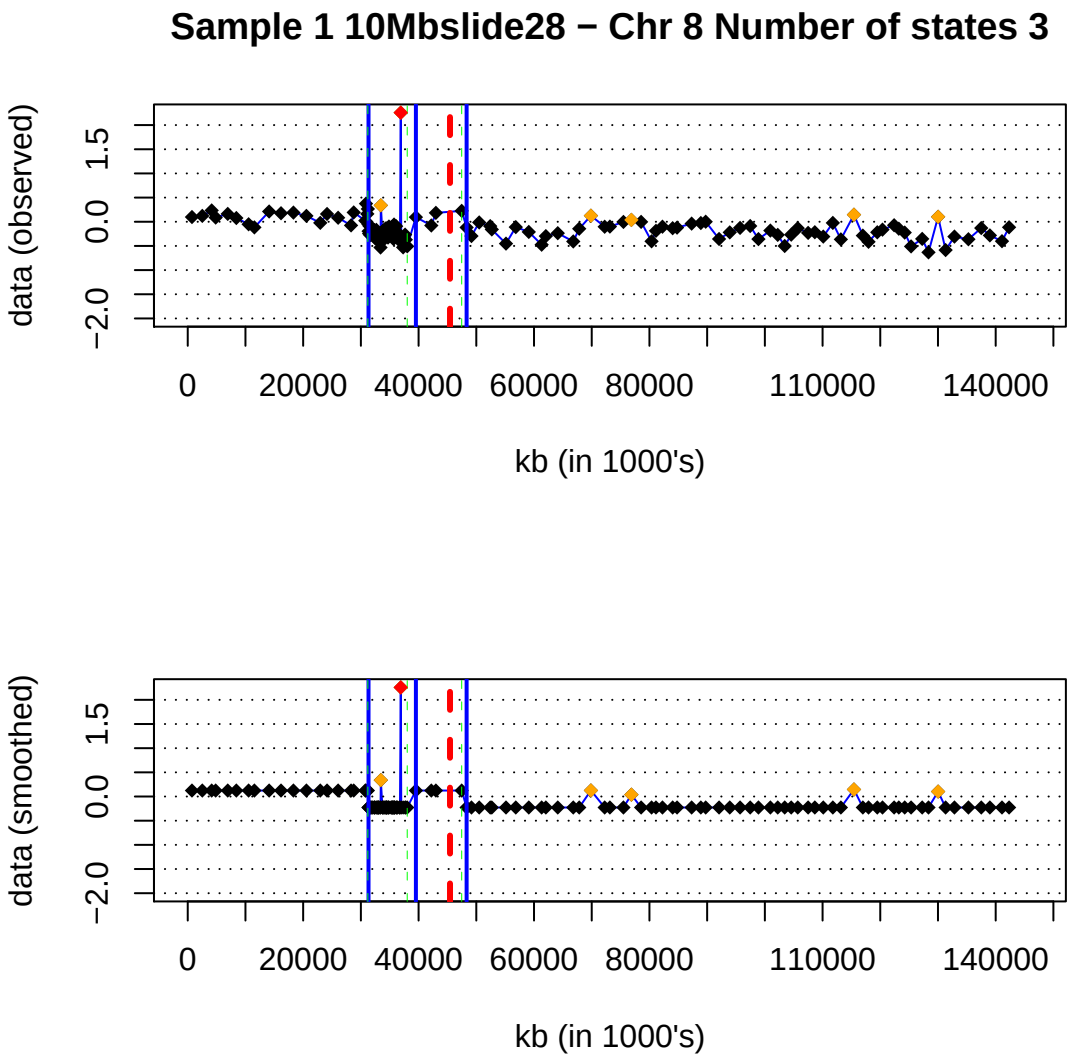
Here we cluster the samples. Again this function will except an object of class `MAList` or `SegList`. We have specified which chromosomes to cluster using the `vecchrom` argument as well as defining the colours that represent areas of amplification or deletion.

```
> clusterGenome(MA2)
```




The `plotSegmentationStates` function provides a visual representation of the observed M-values for a particular chromosome alongside the predicted values produced by the segementation algorithm. It requires both a `SegList` and a `GenList` object. The `sample.ind` argument specifies which array(s) we would like to be plotted, whilst the desired chromosome(s) are passed via the `chr` parameter.

```
> plotSegmentationStates(SegInfo.merged, GenList, array = 1,
+     chr = 8)
```


Sample 1 10Mbslide28 − Chr 8 Number of states 3
data (observed)
−2.0
0.0
1.5
0
20000
40000
60000
80000
110000
140000
kb (in 1000's)
data (smoothed)
−2.0
0.0
1.5
0
20000
40000
60000
80000
110000
140000
kb (in 1000's)