Sample of aCGH analysis

ML Smith

February 13, 2006

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. In addition to *limma*, the following packages are also loaded: *GLAD*, *DNAcopy* and *aCGH*. Each of these impliments an alternative segmentation method that may be applied to the data.

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")
```

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 Mb) 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)
```

Commonly, when aCGH experiments are carried out the reference channel is dyed using Cy5 and the test channel is dyed using Cy3. This is the opposite way to expression data. In order to take this into account we need to specify which channel is the reference within our *RGList*. To do this we create a design vector with each column corresponding to an array in the experiment. A value of 1 indicates that the Cy3 channel is the reference, whilst a value of -1 equates to Cy5 being the reference.

```
> RG1$design <- c(-1, -1)
```

We now proceed to use the function `backgroundCorrect` to remove the background intensity for each spot. In this example we have chosen the method 'minimum' which subtracts the background value from the forground. Any intensity which is zero or negative after the background subtraction is set equal to half the minimum of the positive corrected intensities for that array. For other background correction methods please see the appropriate help file.

```
> 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 `processCGH` to 'tidy up' the MAList object. The *method.of.averaging* option defines how clones of the same type should be averaged. If this is specified the duplicates are removed following the averaging leaving only one occurence of each clone set.

```
> MA2 <- processCGH(MA, method.of.averaging = mean)
```

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 `runDNAcopy` and `runGLAD` for details of how to use them.

```
> SegInfo.Hom <- runHomHMM(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.

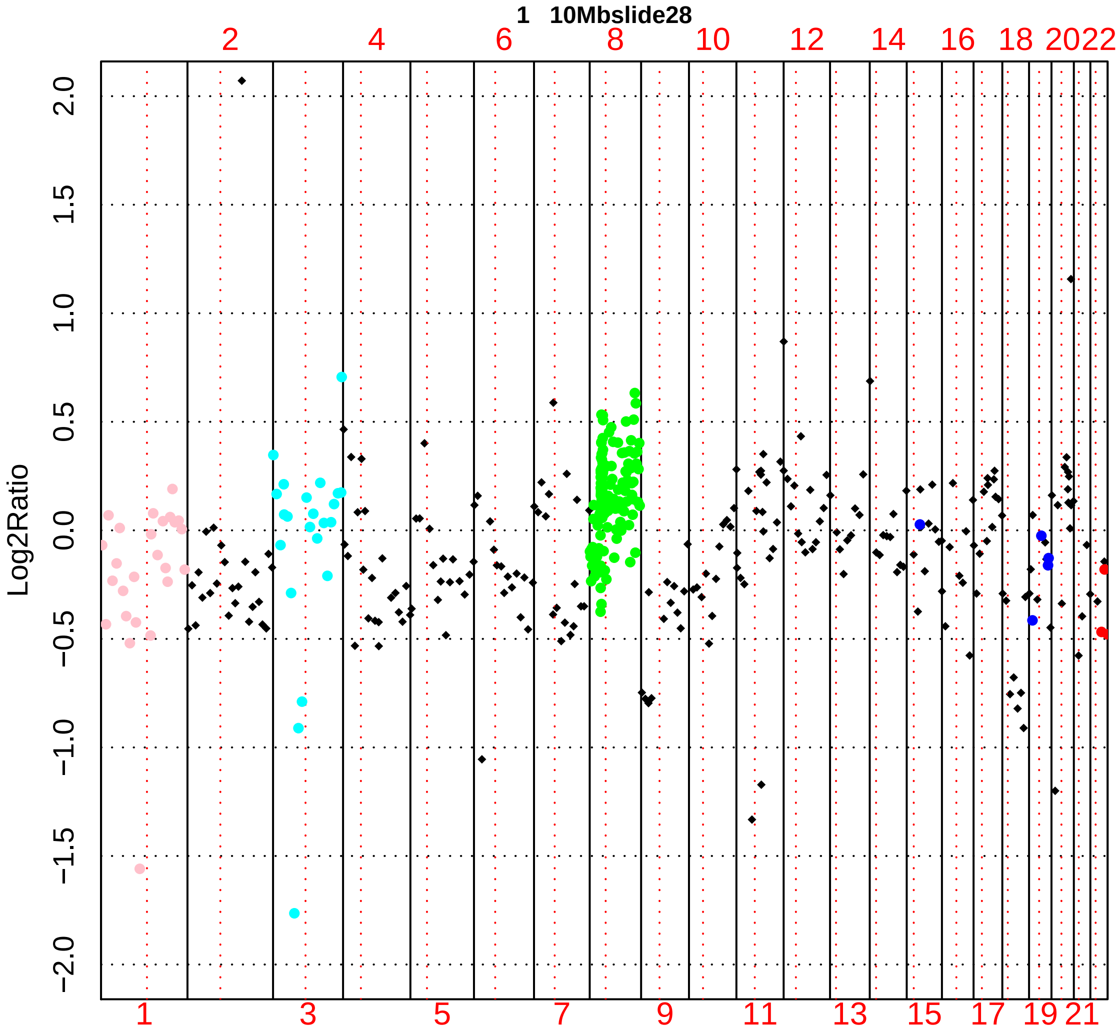
```
> SegInfo.Hom.merged <- mergeStates(SegInfo.Hom, MergeType = 1)
```

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

3 Plotting Functions

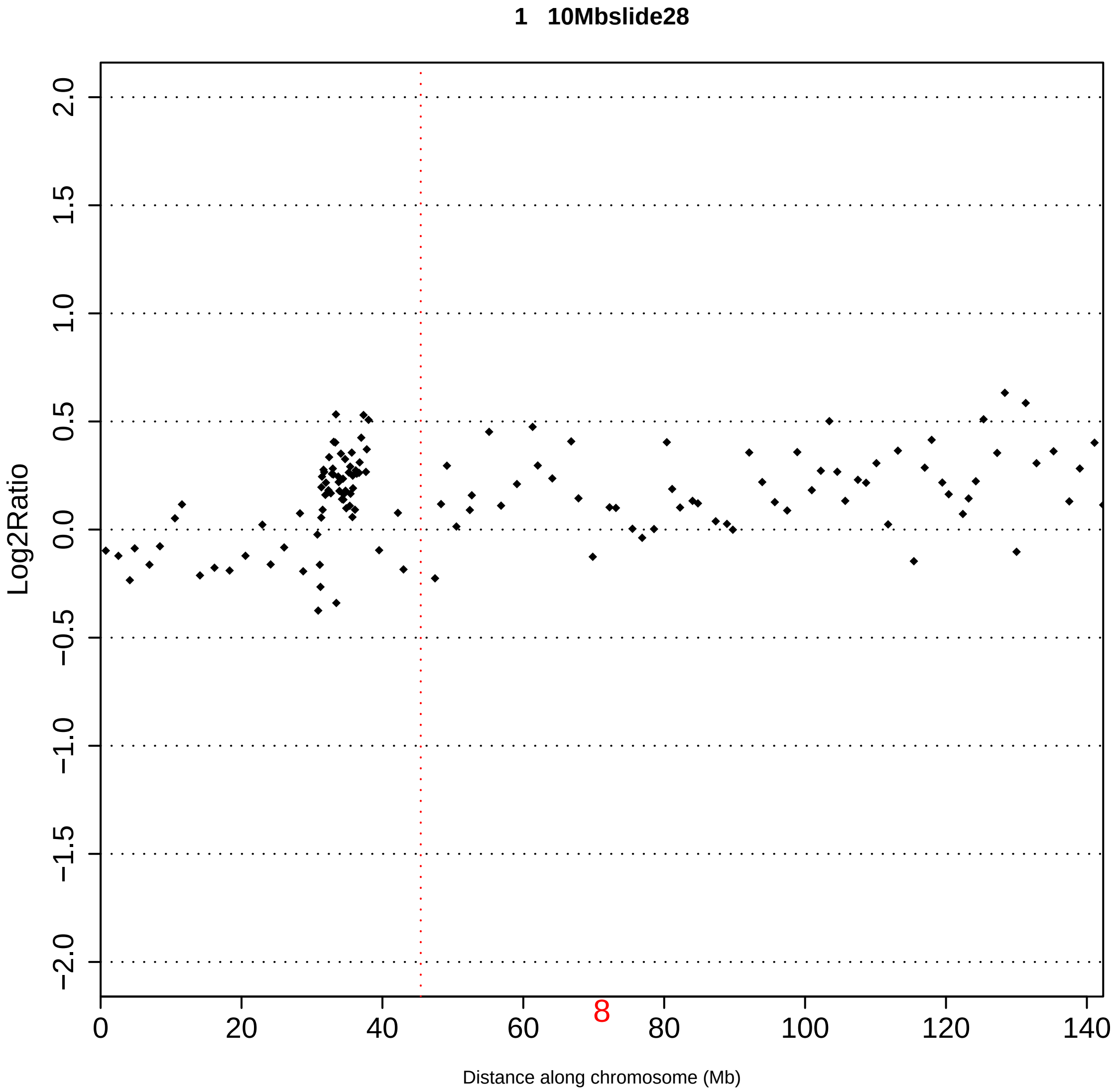
The library comes with a variety of plotting functions that provide visual representations of the data at various stages of the analysis process. Firstly we will look at the `genomePlot` function. This function takes either an *MAList* or a *SegList* object (in this example we've used an *MAList*) and 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.

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


1 10Mbslide28
Log2Ratio
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

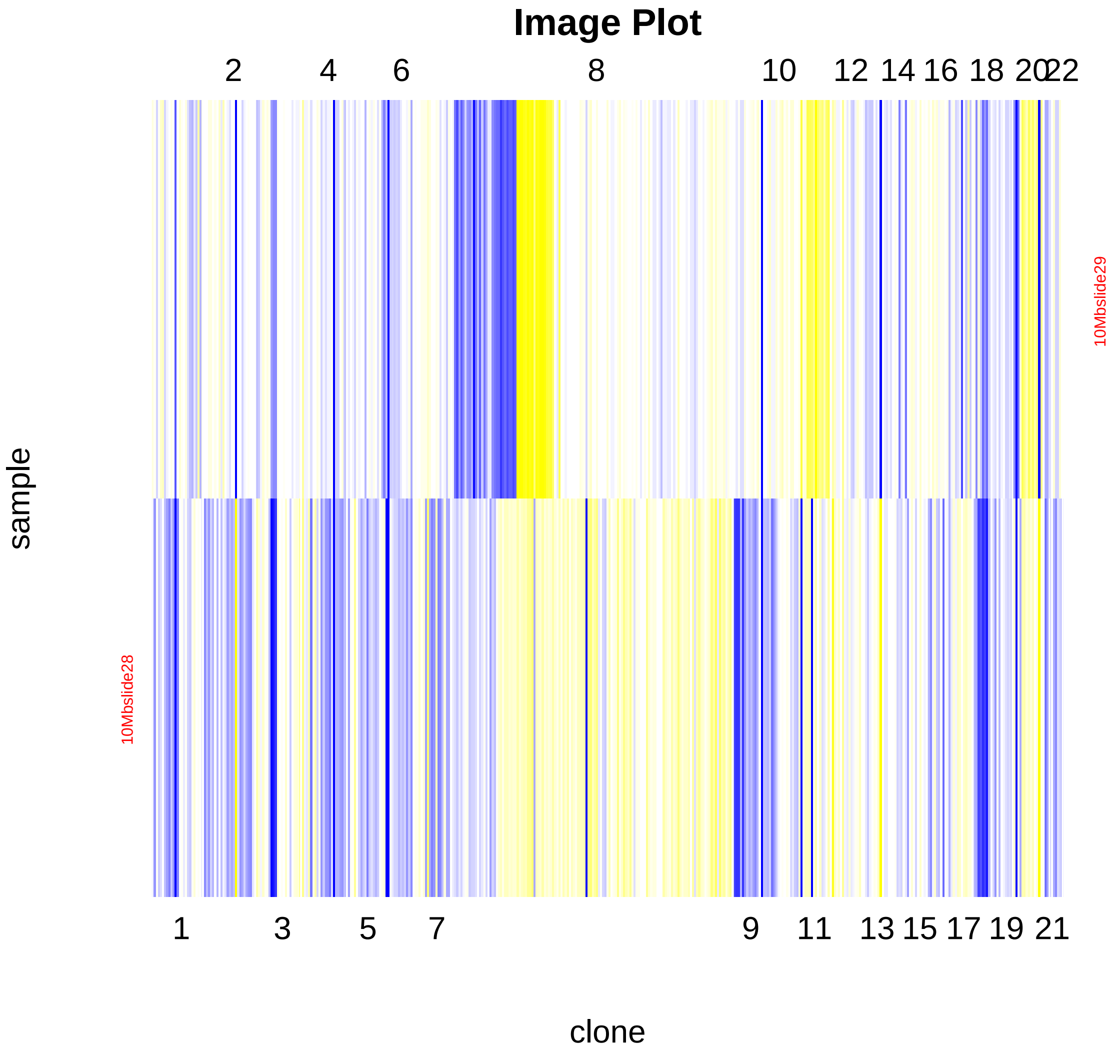
It is also possible to look at specific chromosomes, rather than the entire genome as in the previous example. Which particular chromosome is to be plotted is specified using the `chrom.to.plot` argument.

```
> genomePlot(MA2, array = 1, chrom.to.plot = 8)
```



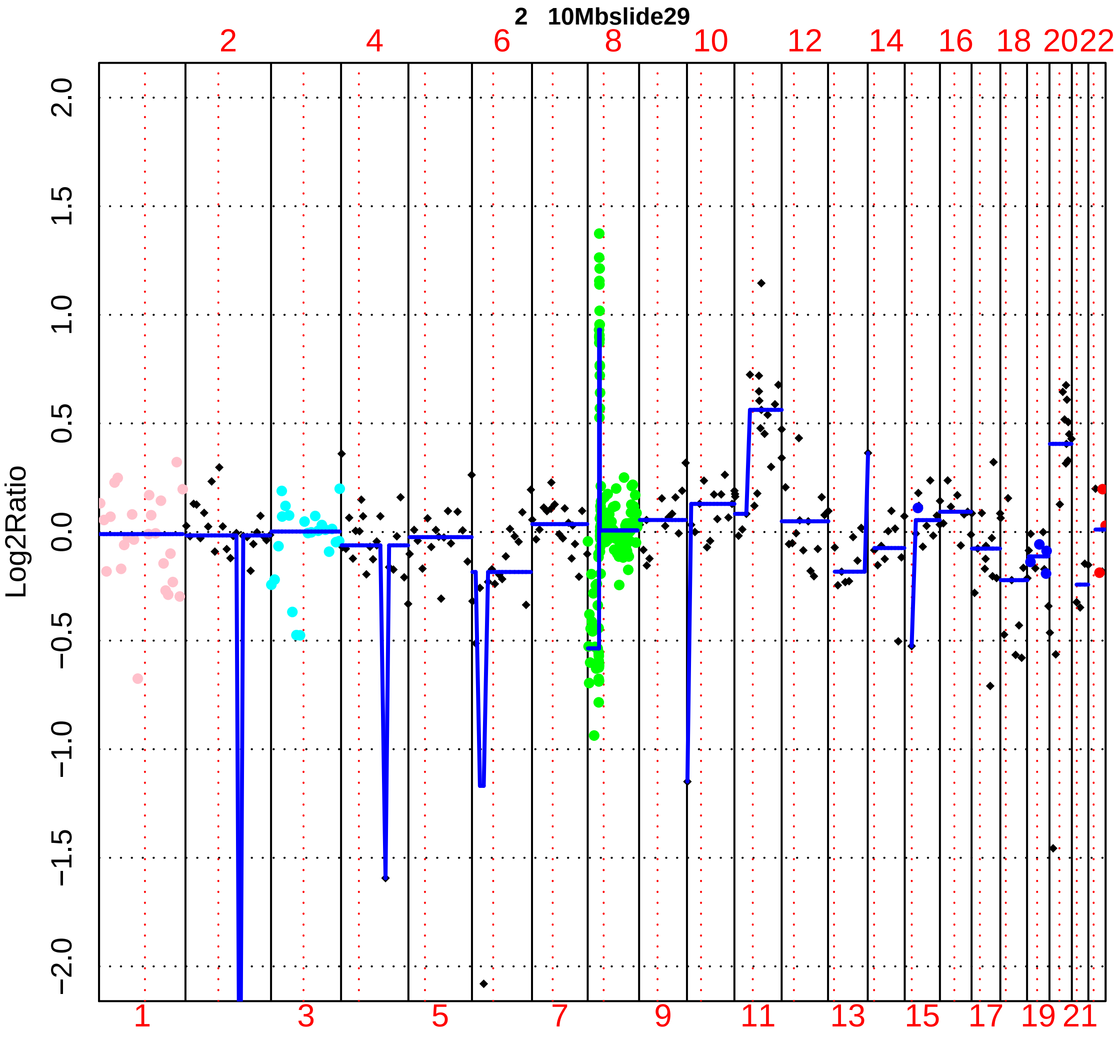
Here we cluster the samples. Again this function will except an object of class `MAList` or `SegList`. It is possible to specify which chromosomes to cluster using the `vecchrom` argument as well as defining the colours that represent areas of amplification or deletion. Please see the help file for more details.

```
> heatmapGenome(MA2)
```



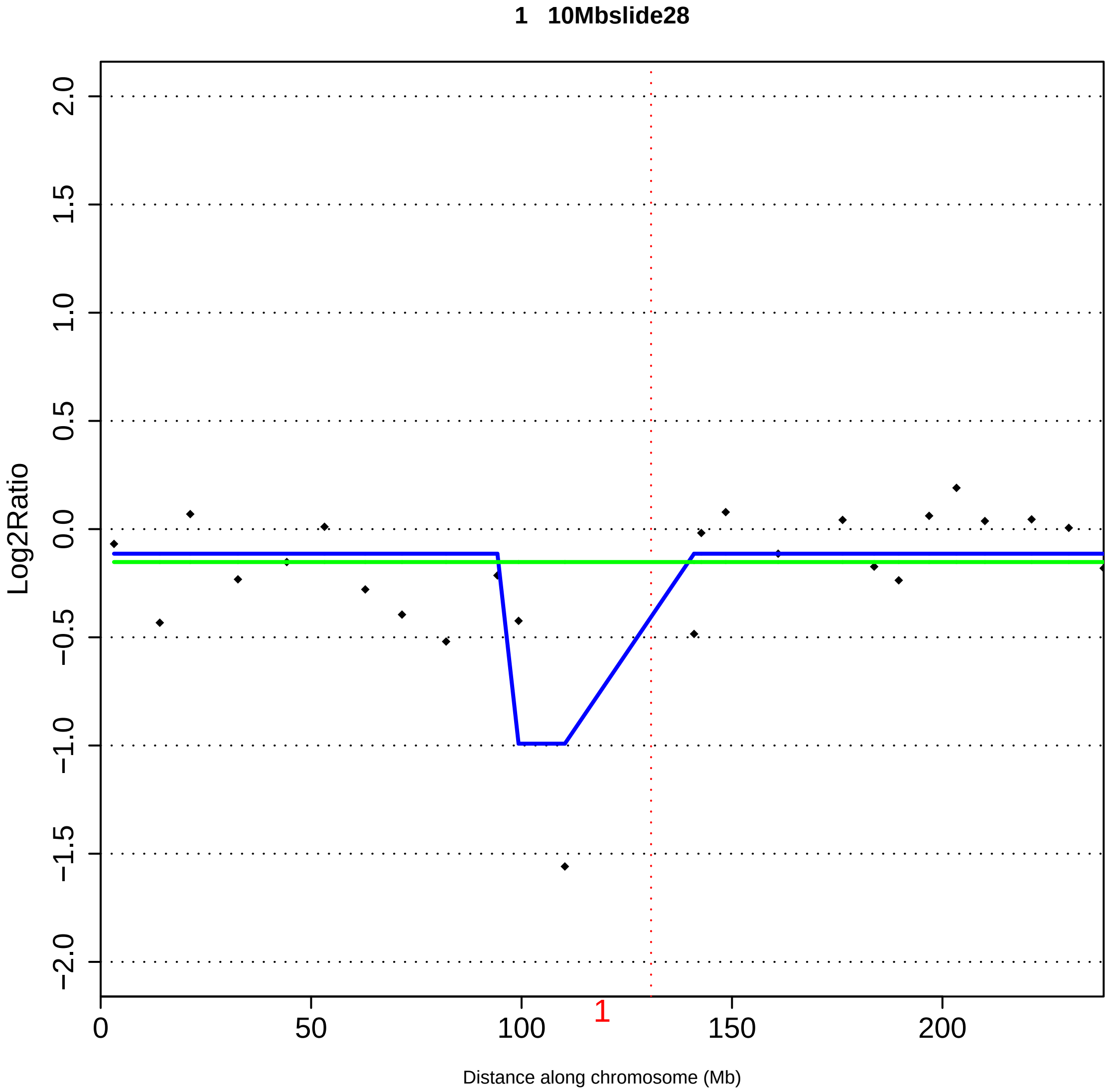
The `plotSegmentedGenome` function provides a visual representation of the observed M-values overlayed with the predicted states produced by the segementation algorithm. It requires a `SegList` as input.

```
> plotSegmentedGenome(SegInfo.Hom.merged, array = 2)
```



Using the argument *chrom.to.plot* it is possible to specify individual chromosomes to plot. Additionally the function can accept more than one `SegList` allowing visual comparison between segmentation methods.

The following example applies the DNAcopy algorithm to the data, merges it and then plots both that segmentation method and the homogeneous HMM on the same axis.

```
> Seg.DNAcopy <- runDNAcopy(MA2)
> SegInfo.DNAcopy.merged <- mergeStates(Seg.DNAcopy)
> plotSegmentedGenome(SegInfo.Hom.merged, SegInfo.DNAcopy.merged,
+     array = 1, chrom.to.plot = 1, colors = c("blue", "green"))
```