# snapCGH: Segmentation, Normalization and Processing of aCGH Data Users' Guide

ML Smith, JC Marioni, TJ Hardcastle, NP Thorne

April 25, 2007

## Citing snapCGH

If you have used *snapCGH* in your work please cite the package using the following:

> Smith, M.L., Marioni, J.C., Hardcastle, T.J., Thorne, N.P.
> snapCGH: Segmentation, Normalization and Processing of aCGH Data Users' Guide, Bioconductor, 2006

If you make use of the function `runBioHMM`, please cite:

> Marioni, J. C., Thorne, N. P., and Tavaré, S. (2006).
> BioHMM: a heterogeneous hidden Markov model for segmenting array CGH data.
> *Bioinformatics* **22**, 1144 - 1146

## Introduction

This document outlines some of the commands used to read in, investigate and subsequently segment array CGH data. The files analysed represent 2 breast cancer cell lines obtained from Jessica M Pole and Paul AW Edwards.

```
> 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*, *tilingArray* and *aCGH*. Each of these impliments an alternative segmentation method that may be applied to the data.

## 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")
```

Positional information about the clones on the array (e.g. which chromosome and the position on the chromosome a clone if from) can be incorporated using the `readPositionalInfo` function. This function accepts either an `RGList` or `MAList` along with an argument specifying which array platform the data was produced on. Currently the only supported platforms are Agilent, Bluefuse and Nimblegen, but this should expand in the near future.

If your platform isn't supported or `readPostitionalInfo` returns an error (e.g. if the a new version of the array output data becomes incompatible with the current function) then the positional information 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. The X and Y chromosomes can be labelled either as X and Y or 23 and 24. Both instances are handled by `read.clonesinfo`. The clones info file must be ordered in the same way as the clones are ordered in the array output data.

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, as is the case in this example.

```
> 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. If you feel that no normalisation is required then it is possible to skip this stage and proceed directly onto the next step maintaining the *RGList*.

```
> 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. The *ID* argument is used to indicate the name of the column containing a unique identifier for each clone type. It is this identifier that is used when averaging replicates. As mentioned above this function accepts both *RGLists* and *MALists*.

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

## Segmentation

We are now ready to fit the segmentation method. For larger data sets this step can take a long time (several hours). In this example we call the homogeneous hidden Markov model available in the *aCGH* package.

```
> SegInfo.Hom <- runHomHMM(MA2, criteria = "AIC")
```

At the current time there are methods for calling four other segmentation algorithms in addition to the method shown above. Three of these are included as wrapper functions to methods available in other packages, specifically: *DNAcopy*, *GLAD* and *tilingArray*. These are respectively called in the following ways:

```
> SegInfo.DNAcopy <- runDNAcopy(MA2)
> SegInfo.GLAD <- runGLAD(MA2)
> SegInfo.picard <- runPicard(MA2)
```

The final method, called BioHMM, is a heterogeneous hidden Markov model and is maintained within *snapCGH*. It can be called using the following command. By default it incorporates the distance between clones into the model assigning a higher probability of state change to clones that are a larger distance apart on a chromosome. This option is control using the argument *useCloneDists*.

```
> SegInfo.Bio <- runBioHMM(MA2)
```

We now deal with the fact that the segmentation methods sometimes have 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.

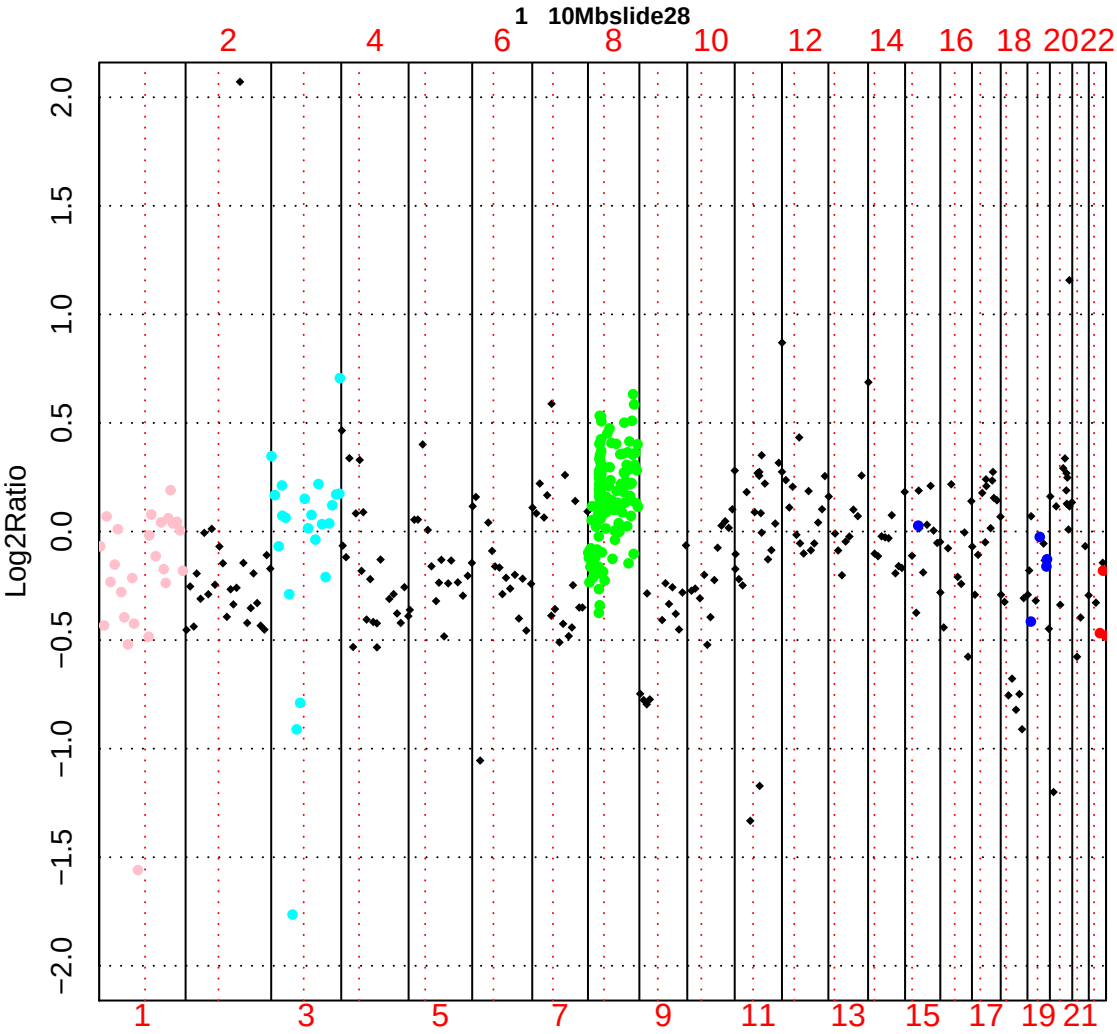
```
> SegInfo.Hom.merged <- mergeStates(SegInfo.Hom, MergeType = 1)
```

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

## Plotting Functions

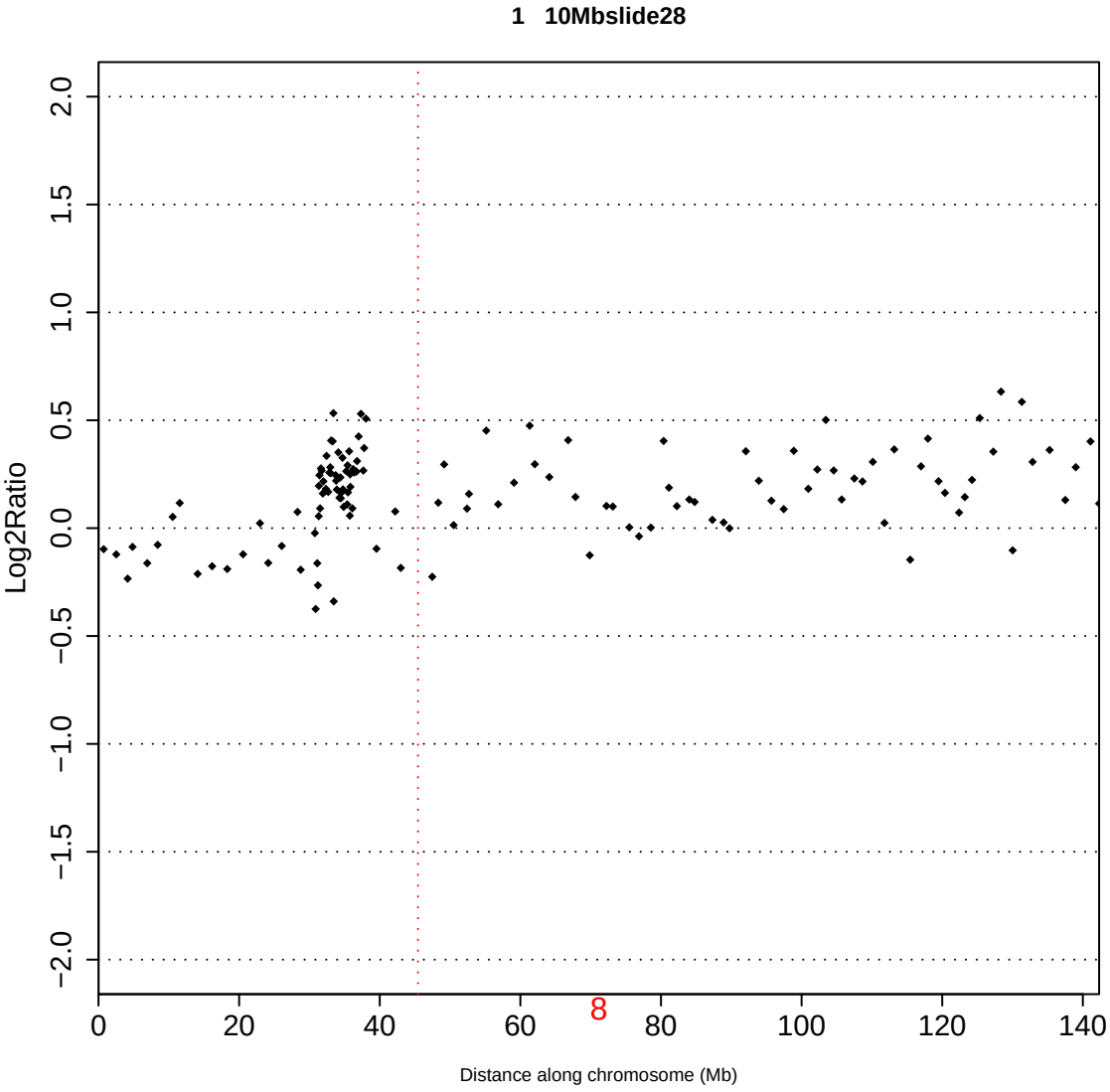
The library comes with a variety of plotting functions that provide visual representations of the data at various stages of the analysis process. Firstly we will look at the `genomePlot` function. This function takes either an *MAList* or a *SegList* object (in this example we've used an *MAList*) and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



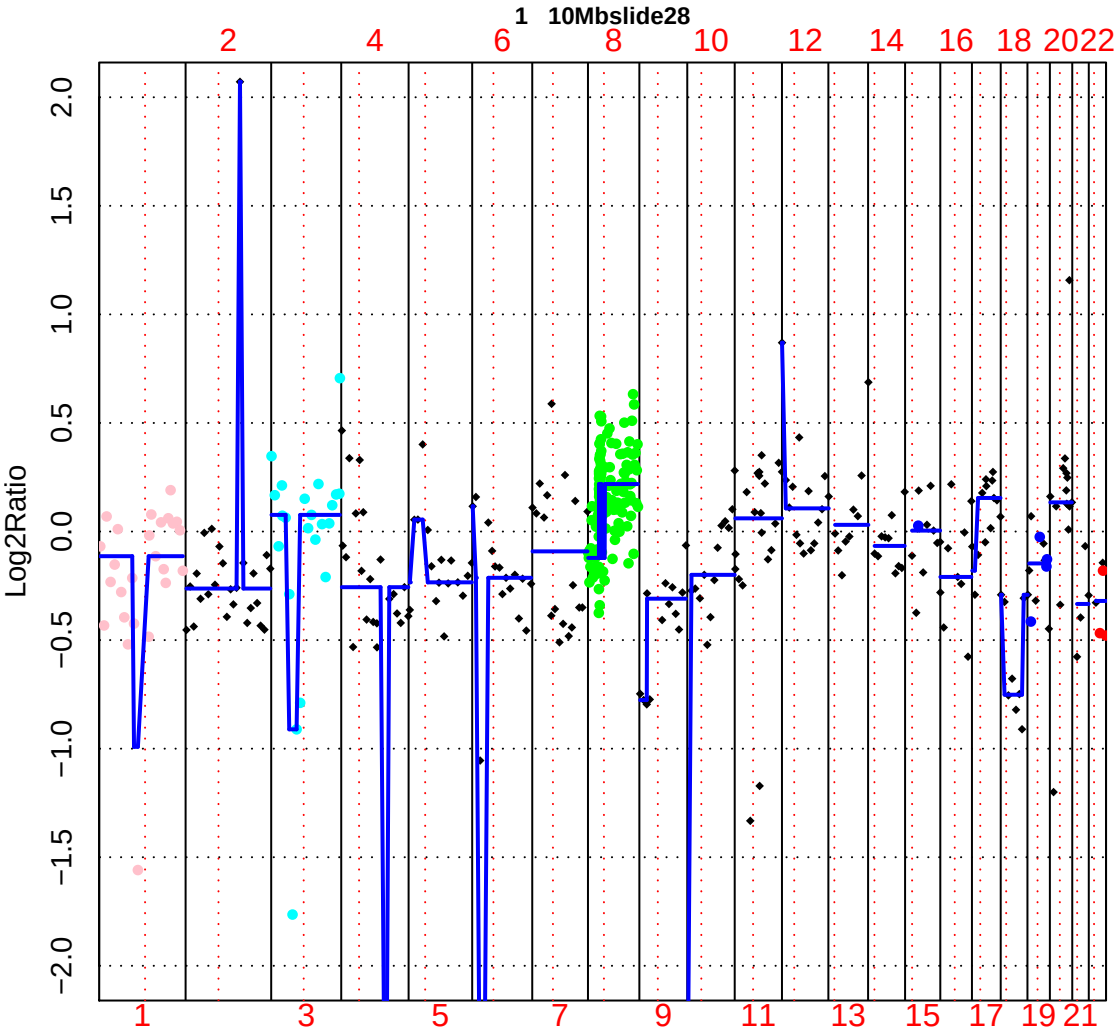

It is also possible to look at specific chromosomes, rather than the entire genome as in the previous example. Which particular chromosome is to be plotted is specified using the *chrom.to.plot* argument.

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




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 = 1)
```

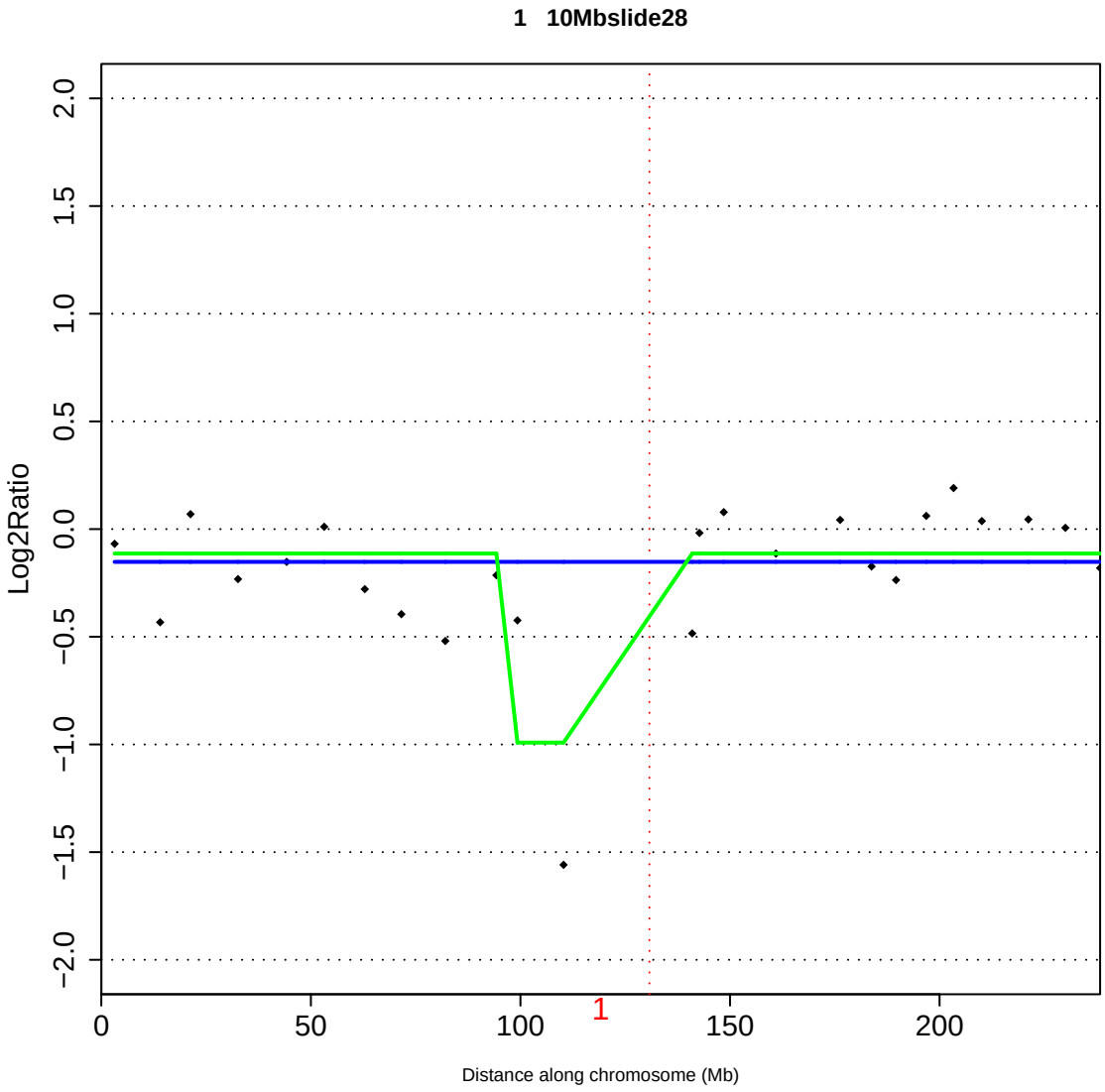



Using the argument *chrom.to.plot* it is possible to specify individual chromosomes to plot. Additionally the function can accept more than one `SegList` allowing visual comparison between segmentation methods.

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, coloring them blue and green respectively.

```
> Seg.DNAcopy <- runDNAcopy(MA2)
> SegInfo.DNAcopy.merged <- mergeStates(Seg.DNAcopy)
> plotSegmentedGenome(SegInfo.DNAcopy.merged, SegInfo.Hom.merged,
```

```
+     array = 1, chrom.to.plot = 1, colors = c("blue", "green"))
```



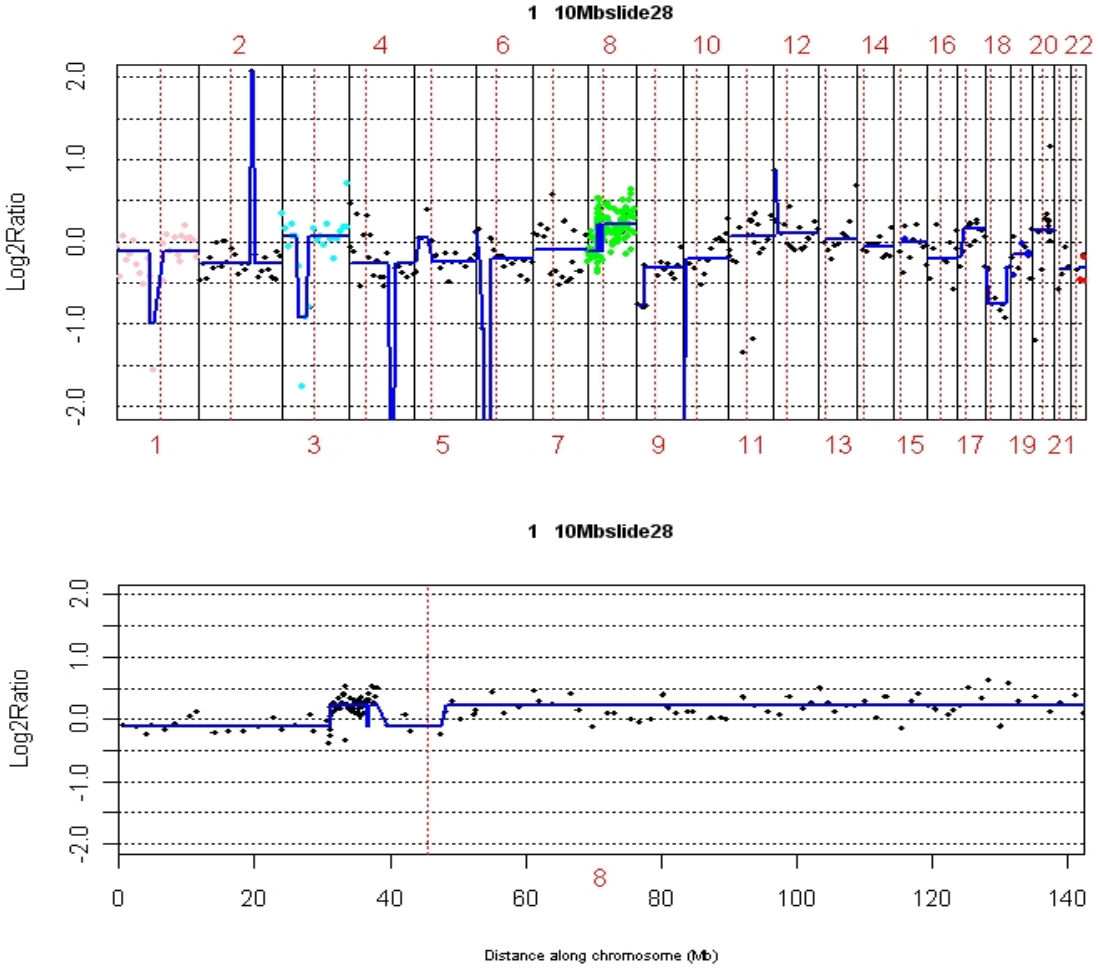
## Interactive Plotting Functions

*snapCGH* includes several interactive plotting functions to allow users to view data on a large scale and the focus in on particular areas of intrest.

The first of these is `zoomGenome`. This function splits the screen in two horizontally and plots the same as `plotSegmentedGenome` in the upper half. It is then possible to click on any of the chromosomes displayed and the selected chromosome is plotted in the lower half of the graphics window. Clicking to either side of the plot will end the interactivity.

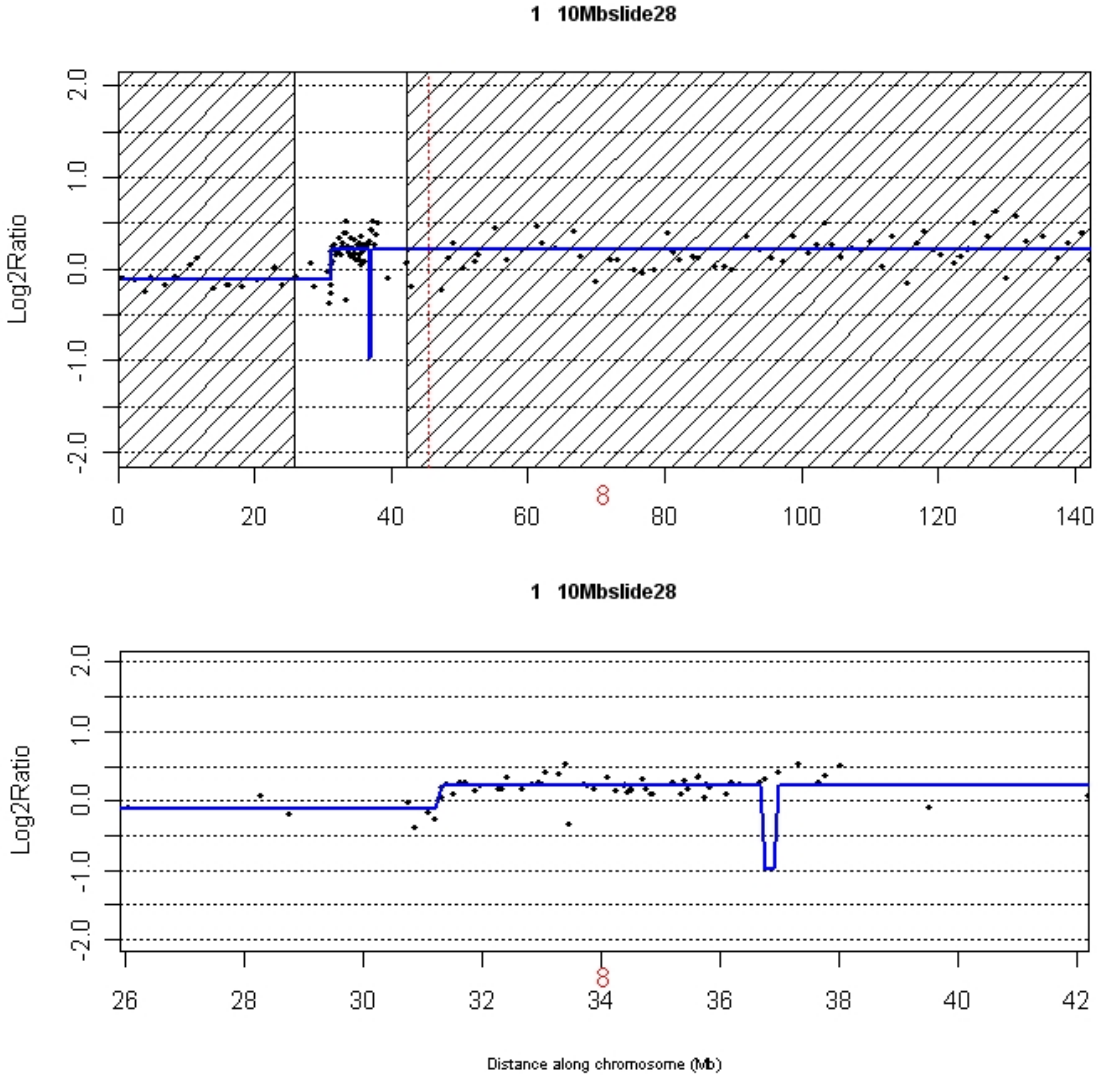
In the example below we have plotted the first array in the `SegInfo.Hom.merged` object and then clicked on chromosome eight.

```
> zoomGenome(SegInfo.Hom.merged, array = 1)
```



`zoomChromosome` is an equivalent function for focusing on specific areas in a specific chromosome. The layout is the same as in the `zoomGenome` but this time the upper plot is of a specified chromosome. Click in two location on this plot and the selected region of the chromosome will be plotted below. The region of focus is also indicated in the upper plot. Clicking twice again will change the focus region. This function accepts either an *MAList* or multiple *SegList* objects.

```
> zoomChromosome(SegInfo.Hom.merged, array = 1, chrom.to.plot = 8)
```



## Simulating Data

Simulated data is often used to assess the efficacy of segmentation methods so *snapCGH* also provides facilities for simulating aCGH data. None of the currently available simulation schemes take into account the spatial nature of aCGH data, they simply generate ordered $log_2$ ratios. However with the development2 of new segmentation methods, such as BioHMM, new simulation schemes are needed. In particular the simulation scheme available in *snapCGH* allows differentiation between tiled and non-tiled regions to emulated different array technologies.

The function `simulateData` generates a *SegList* of 22 chromosomes with the number of arrays specified by the argument *nArrays*. The function has a mulitude of additional arguments for specifying the specific type of data you wish to simulate. These are documented in the appropriate helpfile.

```
> simulation <- simulateData(nArrays = 4)
```

In the following example we run the homogeneous HMM and the DNAcopy algorithm on the simulated data. The function `compareSegmentations` can be used to evaluate the performance of a segmentation methods against the known truth of the simulated data. It returns true positive and false discovery rates for breakpoint detection in each supplied *SegList*. The argument *offset*, which can take an integer between 0 and 2, specifies how close the algorithm needs to be to a real breakpoint before it is scored.

```
> Sim.HomHMM <- runHomHMM(simulation)
> Sim.DNAcopy <- runDNAcopy(simulation)
> rates <- compareSegmentations(simulation, offset = 0, Sim.HomHMM,
+     Sim.DNAcopy)
```

The output from `compareSegmentations` is a list containing two matrices. The first of these, `$TPR`, contains the true positive rate, whilst the second, `$FDR`, holds the false discovery rate. Both of these matrices are arranged such that a row represents a segmentation method and each column is an array.

```
> rates

$TPR
          Sample 1 Sample 2  Sample 3  Sample 4
HomHMM  0.4716981 0.484375 0.4590164 0.4363636
DNAcopy 0.6981132 0.718750 0.6721311 0.6727273

$FDR
          Sample 1  Sample 2  Sample 3   Sample 4
HomHMM  0.4897959 0.5373134 0.5625000 0.52000000
DNAcopy 0.1777778 0.0800000 0.1632653 0.09756098
```

Shown below are boxplots of the `rates` object.

```
> par(mfrow = c(1, 2))
> boxplot(rates$TPR ~ row(rates$TPR), col = c("red", "blue"), main = "True Positive Rate")
> boxplot(rates$FDR ~ row(rates$FDR), col = c("red", "blue"), main = "False Discovery Rate")
```