

# An Introduction to the “genoset” Package

Peter M. Haverty

April 15, 2011

## 1 Introduction

The `genoset` package offers an extension of the BioConductor `eSet` object for genome arrays. The package offers three classes. The first class is the “`GenoSet`” class which can hold an arbitrary number of equal-sized matrices in its `assayData` slot. The principal addition of the `GenoSet` class is a “`locData`” slot that holds a `RangedData` object from the `IRanges` package. The `locData` slot allows for quick subsetting by genome position.

Two classes extend `GenoSet`: `CNSet` and `BAFSet`. `CNSet` is the basic copy number object. It keeps its data in the “`cn`” slot, similar to the “`exprs`” slot of the `ExpressionSet`. `BAFSet` is intended to store “LRR” or Log-R Ratio and “BAF” or B-Allele Frequency data for SNP arrays. LRR and BAF come from the terms coined by Illumina. LRR is copynumber data processed on a per-snp basis to remove some variability using the expected log-ratio of normal samples with the same genotype. BAF represents the fraction of signal coming from the “B” allele, relative to the “A” allele, where A and B are arbitrarily assigned. BAF has the expected value of 0 or 1 for HOM alleles and 0.5 for HET alleles. Deviation from these expected values can be interpreted as Allelic Imbalance, which is a sign of gain, loss, or copy-neutral LOH.

I will use a `CNSet` to demonstrate the features that apply to both classes. Some `BAFSet`-specific features will be handled later in the document.

### 1.1 Creating Objects

`GenoSet`, `CNSet` and `BAFSet` objects can be created using the functions with the same name. Let’s load up some fake data to experiment with. Don’t worry too much about how the fake data gets made.

Notice how `CNSet` requires a ‘`cn`’ element, `BAFSet` requires ‘`lrr`’ and ‘`baf`’, and `genoset` can take `assayData` elements with any name. Also notice how `assayData` elements can be matrices or `DataFrames` with `Rle` columns.

```
> library(genoset)
> data(genoset)
> genoset.ds = GenoSet(locData = locData.rd, foo = fake.lrr, pData = fake.pData,
+   annotation = "SNP6")
> baf.ds = BAFSet(locData = locData.rd, lrr = fake.lrr, baf = fake.baf,
+   pData = fake.pData, annotation = "SNP6")
> cn.ds = CNSet(locData = locData.rd, cn = fake.lrr, pData = fake.pData,
+   annotation = "SNP6")
> cn.ds
```

```
CNSet (storageMode: lockedEnvironment)
assayData: 1000 features, 3 samples
  element names: cn
protocolData: none
```

```

phenoData
  sampleNames: K L M
  varLabels: a b ... e (5 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: SNP6
Feature Locations:
RangedData with 1000 rows and 0 value columns across 3 spaces

```

	space	ranges	
	<factor>	<IRanges>	
p1	chr12	[ 1, 1]	
p2	chr12	[ 325001, 325001]	
p3	chr12	[ 650001, 650001]	
p4	chr12	[ 975001, 975001]	
p5	chr12	[1300001, 1300001]	
p6	chr12	[1625001, 1625001]	
p7	chr12	[1950001, 1950001]	
p8	chr12	[2275001, 2275001]	
p9	chr12	[2600001, 2600001]	
...	...	...	...
p992	chr8	[392000000, 392000000]	
p993	chr8	[393000000, 393000000]	
p994	chr8	[394000000, 394000000]	
p995	chr8	[395000000, 395000000]	
p996	chr8	[396000000, 396000000]	
p997	chr8	[397000000, 397000000]	
p998	chr8	[398000000, 398000000]	
p999	chr8	[399000000, 399000000]	
p1000	chr8	[400000000, 400000000]	

Universe: hg19

```

> rle.baf.ds = BAFSet(locData = locData.rd, lrr = DataFrame(K = Rle(c(rep(1.5,
+ 300), rep(2.3, 700))), L = Rle(c(rep(3.2, 700), rep(2.1, 300))),
+ M = Rle(rep(1.1, 1000)), row.names = rownames(fake.lrr)), baf = DataFrame(K = Rle(c(rep(0.05,
+ 600), rep(0.5, 400))), L = Rle(c(rep(0, 700), rep(0.5, 300))), M = Rle(rep(1,
+ 1000)), row.names = rownames(fake.baf)), pData = fake.pData, annotation = "SNP6")

```

## 1.2 Using the Subset Features

GenoSet objects can be subset using chromosome names, or with a set of ranges

```

> chr12.ds = cn.ds[["chr12"]]
> chr12.ds

```

```

CNSet (storageMode: lockedEnvironment)
assayData: 400 features, 3 samples
  element names: cn
protocolData: none
phenoData
  sampleNames: K L M
  varLabels: a b ... e (5 total)

```

```

varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: SNP6
Feature Locations:
RangedData with 400 rows and 0 value columns across 1 space

```

	space	ranges	
	<factor>	<IRanges>	
p1	chr12	[ 1, 1]	
p2	chr12	[ 325001, 325001]	
p3	chr12	[ 650001, 650001]	
p4	chr12	[ 975001, 975001]	
p5	chr12	[1300001, 1300001]	
p6	chr12	[1625001, 1625001]	
p7	chr12	[1950001, 1950001]	
p8	chr12	[2275001, 2275001]	
p9	chr12	[2600001, 2600001]	
...	...	...	...
p392	chr12	[127075001, 127075001]	
p393	chr12	[127400001, 127400001]	
p394	chr12	[127725001, 127725001]	
p395	chr12	[128050001, 128050001]	
p396	chr12	[128375001, 128375001]	
p397	chr12	[128700001, 128700001]	
p398	chr12	[129025001, 129025001]	
p399	chr12	[129350001, 129350001]	
p400	chr12	[129675001, 129675001]	

```
Universe: hg19
```

```

> gene.rd = RangedData(ranges = IRanges(start = c(3.5e+07, 1.27e+08),
+   end = c(35500000, 1.29e+08), names = c("HER2", "CMYC")), space = c("chr17",
+   "chr8"))
> gene.ds = cn.ds[gene.rd, ]
> gene.ds

```

```

CNSet (storageMode: lockedEnvironment)
assayData: 5 features, 3 samples
  element names: cn
protocolData: none
phenoData
  sampleNames: K L M
  varLabels: a b ... e (5 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: SNP6
Feature Locations:
RangedData with 5 rows and 0 value columns across 2 spaces

```

	space	ranges	
	<factor>	<IRanges>	
p515	chr17	[ 35200000, 35200000]	
p516	chr17	[ 35500000, 35500000]	

```
p727      chr8 [127000000, 127000000] |
p728      chr8 [128000000, 128000000] |
p729      chr8 [129000000, 129000000] |
Universe: hg19
```

GenoSet objects can also be subset by a group of samples and/or features, just like an ExpressionSet.

```
> cn.ds[1:4, 1:2]

CNSet (storageMode: lockedEnvironment)
assayData: 4 features, 2 samples
  element names: cn
protocolData: none
phenoData
  sampleNames: K L
  varLabels: a b ... e (5 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: SNP6
Feature Locations:
RangedData with 4 rows and 0 value columns across 1 space
      space      ranges |
  <factor>    <IRanges> |
p1   chr12 [    1,    1] |
p2   chr12 [325001, 325001] |
p3   chr12 [650001, 650001] |
p4   chr12 [975001, 975001] |
Universe: hg19
```

### 1.3 Slots

CNSet objects extend ExpressionSet objects and so have all of the same functions and slots, except that the exprs slot and getter and setter functions have been replaced with “cn”. The locData slot has been added. This slot holds a RangedData object that keeps track of the feature positions and allows for quick subsetting. The RangedData object is always sorted such that the chromosomes are in contiguous blocks, which provides a constraint on the ordering of the CNSet. There are a number of functions for getting portions of the locData data. “chr” and “pos” return the chromosome and position information for each feature. “genoPos” is like “pos”, but it returns the base positions counting from the first base in the genome, with the chromosomes in order by number and then alphabetically for the letter chromosomes. “orderedChrs” returns this ordered vector of unique chromosome names. “chrInfo” returns the genoPos of the first and last feature on each chromosome in addition to the offset of the first feature from the start of the genome. “chrInfo” results are used for drawing chromosome boundaries on genome-scale plots. “pos” and “genoPos” are defined as the floor of the average of each features start and end positions.

### 1.4 Processing Data

GenoSet contains a convenience function for segmenting data for each sample/chr using the DNACopy package (CBS)

```
> assayDataElement(cn.ds, "cn.segs") = runCBS(cn(cn.ds), locData(cn.ds),
+       n.cores = 3)
```

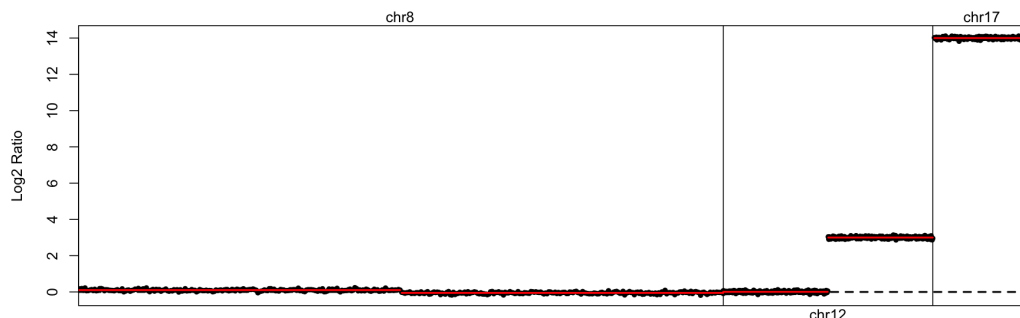


Figure 1: Segmented copy number across the genome for 1st sample

```
Working on segmentation for sample number 1 : K
Working on segmentation for sample number 2 : L
Working on segmentation for sample number 3 : M
```

This function makes use of the multicore package to run things in parallel, so plan ahead when picking “ncores”. Memory usage can be a bit hard to predict.

By default runCBS returns a DataFrame (from the IRanges package) with Run Length Encoded (Rle objects from the IRanges package) columns. This dramatically reduces the amount of memory required. DataFrame objects are valid assayData elements in GenoSet (see object creation above). runCBS can optionally return a list of data.frames, one for each sample. These data.frames are simply the output of DNACopy.

```
> segs = runCBS(cn(cn.ds), locData(cn.ds), n.cores = 3, return.segs = TRUE)
```

These segments can also be obtained from the DataFrame of Rle objects:

```
> segTable(assayDataElement(cn.ds, "cn.segs"), locData(cn.ds))
```

## 1.5 Plots

CNSet has some handy plots for plotting data along the genome.

```
> genoPlot(cn.ds, 1)
> genoPlot(cn.ds, 1, element = "cn.segs", add = TRUE, col = "red")
```

The result is shown in Fig. 1.

```
> genoPlot(cn.ds, 1, chr = "chr12")
> genoPlot(cn.ds, 1, chr = "chr12", element = "cn.segs", add = TRUE, col = "red")
```

The result is shown in Fig. 2.

## 2 BAFSet Objects

BAFSet objects are similar in most ways to CNSet objects, but require at least two types of data, LRR and BAF, in AssayData.

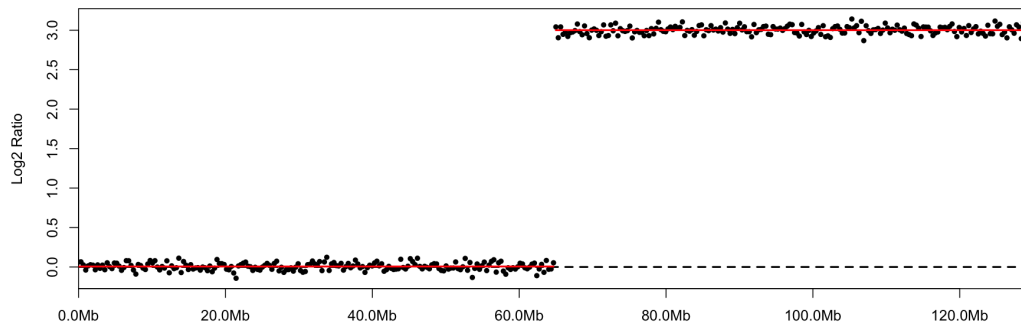


Figure 2: Segmented copy number across chromosome 12 for 1st sample

## 2.1 Processing Data

BAF data can be converted into the “Modified BAF” or mBAF metric, introduced by Staaf, et al., 2008. mBAF folds the values around the 0.5 axis and makes the HOM positions NA. The preferred way to identify HOMs is to use genotype calls from a matched normal (-1,0,1,2 for NA, AA, AB, BB), but NA’ing values over a certain value works OK. A `hom.cutoff` of 0.90 is suggested for Affymetrix arrays and 0.95 for Illumina arrays, following Staaf, et al.

```
> assayDataElement(baf.ds, "mbaf") = baf2mbaf(baf(baf.ds), hom.cutoff = 0.9)
```

Using the HOM SNP calls from the matched normal works much better. A matrix of genotypes can be used to set the HOM SNPs to NA. A list of sample names matches the columns of the genotypes to the columns of your baf matrix. The names of the list should match column names in your baf matrix and the values of the list should match the column names in your genotype matrix. If this method is used and some columns in your baf matrix do not have an entry in this list, then those baf columns are cleaned of HOMs using the `hom.cutoff`, as above.

Both mBAF and LRR can and should be segmented

```
> assayDataElement(baf.ds, "baf.segs") = runCBS(assayDataElement(baf.ds,
+ "mbaf"), locData(baf.ds))
```

```
Working on segmentation for sample number 1 : K
Working on segmentation for sample number 2 : L
Working on segmentation for sample number 3 : M
```

```
> assayDataElement(baf.ds, "lrr.segs") = runCBS(lrr(baf.ds), locData(baf.ds))
```

```
Working on segmentation for sample number 1 : K
Working on segmentation for sample number 2 : L
Working on segmentation for sample number 3 : M
```

## 2.2 Plots

BAFSets (and GenoSets) have a `genoPlot` function as well that additionally requires a “element” argument, which can be “lrr” or “baf” or any other element you happen to have put in “assayData”.

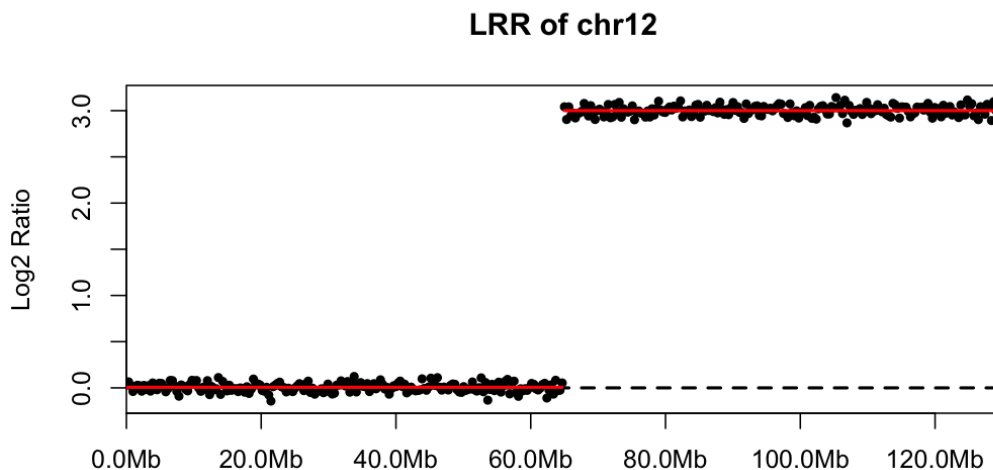


Figure 3: Segmented copy number across the genome for 1st sample

```
> genoPlot(baf.ds, 1, chr = "chr12", element = "lrr", main = "LRR of chr12")
> genoPlot(baf.ds, 1, chr = "chr12", element = "lrr.segs", add = TRUE,
+   col = "red")
```

The result is shown in Fig. 3.

```
> par(mfrow = c(2, 1))
> genoPlot(baf.ds, 1, chr = "chr12", element = "baf", main = "BAF of chr12")
> genoPlot(baf.ds, 1, chr = "chr12", element = "mbaf", main = "mBAF of chr12")
> genoPlot(baf.ds, 1, chr = "chr12", element = "baf.segs", add = TRUE,
+   col = "red")
```

The result is shown in Fig. 4.

## 2.3 Genome Order

Certain methods on `GenoSet` objects expect the rows to be in genome order. In this package, the use of a `RangedData` object for feature location info (the `locData` slot) keeps data for each chromosome in a contiguous block of rows. Users are free to rearrange rows within chromosome as they please. The function `isGenomeOrder` can check if a `rangedData` object, like the `locData` slot, is in genome order. Here “genome order” means that the rows within each chromosome are ordered by start position. “Strict genome order” requires that the chromosome be in a certain order too. The proper order of chromosomes is desirable for full-genome plots and is specified by the `chrOrder` function. The function `genomeOrder` returns a list of integer indices to use to set a `GenoSet` or `RangedData` to weak, or optionally, strict genome order. The object creation methods `GenoSet`, `BAFSet`, and `CNSet` order their objects in strict genome order. Methods, like those in the next section, that require genome order use `genomeOrder` to set it themselves, so users are not required to keep track of whether or not they have changed the row order.

## 2.4 Gene Level Summaries

GenoSet contains feature level data. Often it is desirable to get summaries of assayData matrices over an arbitrary set of ranges, like genes or cytobands. The function “rangeSampleMeans” serves this purpose. Given a RangedData of arbitrary genome ranges, a GenoSet-based object, and the name of a member of that objects assayData slot, “rangeSampleMeans” will return a matrix of values for each of those ranges and for each sample. For each range, “rangeSampleMeans” uses “boundingIndices” to select the features covered by that range and the features bounding that range. The bounding features are the features with locations equal to the start and end of the range or those outside the range and closest to the ends of the range. This bounding ensures that the full extent of the range is accounted for, and more importantly, at least two features are included for each gene, even if the range falls between two features.

As an example, let’s say you want to get the copynumber of your two favorite genes from the subsetting example:

```
> rangeSampleMeans(gene.rd, baf.ds, "lrr")
```

	HER2	CMYC
K	14.0000248	0.09744106
L	-0.5194090	2.99732032
M	0.1941757	0.13363266



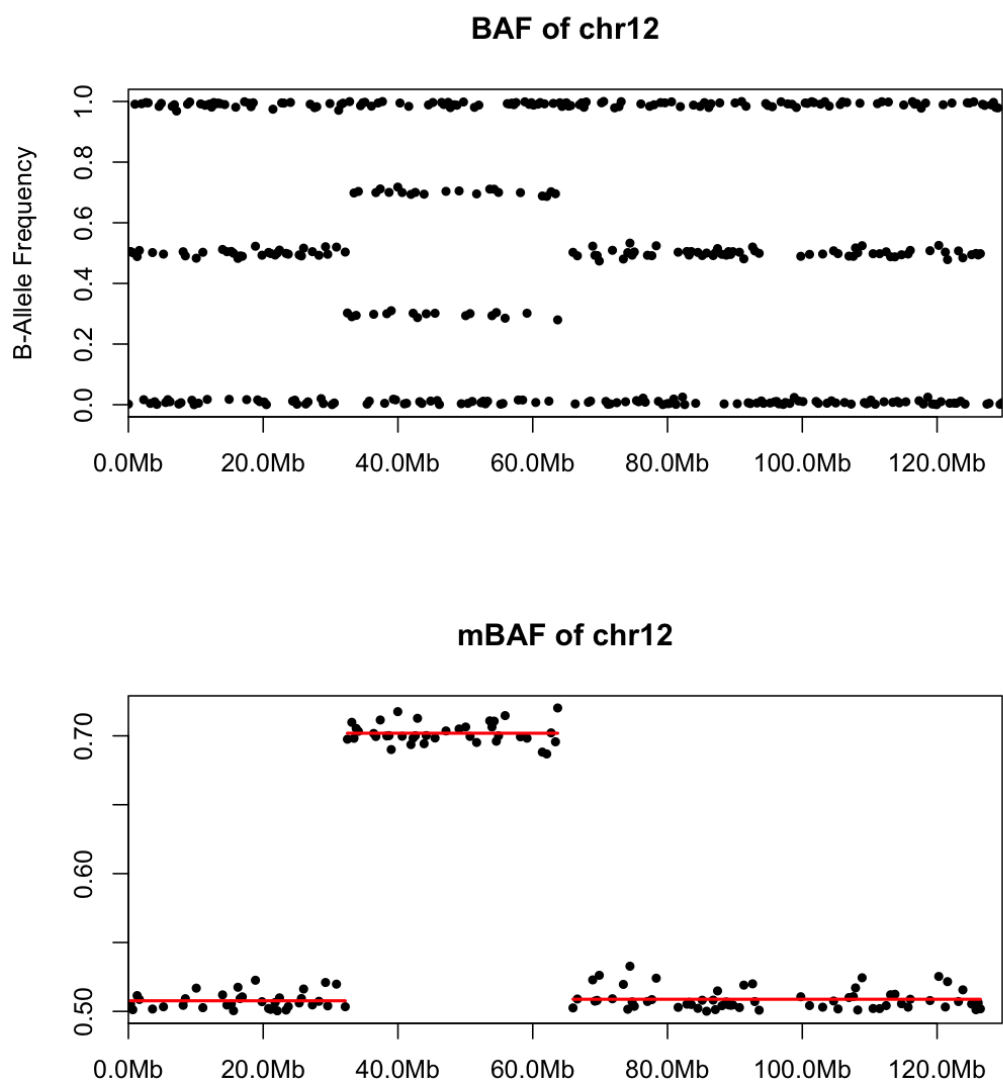


Figure 4: Segmented copy number across the genome for 1st sample