seqCNA: A Package for Copy Number Analysis of High-Throughput Sequencing Cancer DNA

David Mosen-Ansorena[1]

May 3, 2016

[1]Genome Analysis Platform
CIC bioGUNE and CIBERehd
dmosen.gn@cicbiogune.es

Contents

1 Overview

This document guides the reader through the use of the `seqCNA` package. The aim of the package is to process high-throughput sequencing copy number data coming from tumoural samples, departing from SAM or BAM aligned reads up to the final stage of calling the copy numbers. It includes, among other features, an integrated summarization method, several filters based on a range of genomic and read-based characteristics and a normalization approach developed specifically for tumoural data.
Overall, the package was conceived to be user-friendly, provide visual assessment at each step and keep a simple and limited parameterization.

1.1 Example data

The package includes example data that represents the summarized read counts at 200Kbps for chromosomes 1 to 5 of the cancer cell-line HCC1143 (**?**). This output is a subset of what can be achieved by running the summarization function `runSeqsumm` in the package (see `?runSeqsumm`) over the corresponding SAM file, which is too large to be included in the package. Although the example data does not include the sexual chromosomes, the package is able to process them.

1.2 Before you start

Analyses are organized by folder. We recommend placing each SAM or BAM file in a separate folder, where the corresponding analysis files will be generated.

2 Input

An example analysis on the previously described example data follows, beginning by loading the package and the sample data. Note that, with your data, you will need to call `runSeqsumm(file=yourfile.sam)` on your SAM or BAM file instead of just loading it with the function `data`. Also, be sure to have samtools in your path or indicate its location if your file is BAM.

```
> library(seqCNA)
```

```
> data(seqsumm_HCC1143)
```

The first step is to read the data into an R object, which will be updated with new information as we process it.
If we wanted to perform the analysis with a matched normal sample, to indicate a larger summarization window or to use the annotation of a specific genome build, we would indicate it here too.
Here we load the data straight from a dataframe, but the typical use is to indicate the folder where the output generated by `runSeqsumm` is located (seqsumm_out.txt). Notice that no further software is necessary for the summarization except if the input is in BAM format. Then, `runSeqsumm` needs to be told the path to the `samtools` program (`?`).

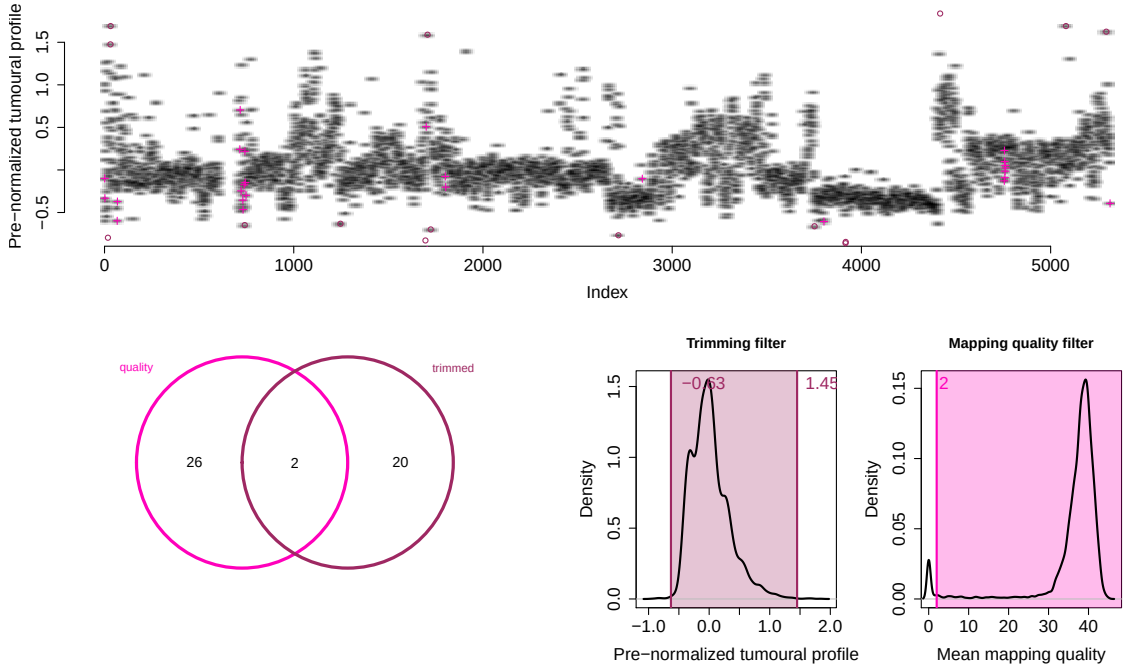
```
> rco = readSeqsumm(tumour.data=seqsumm_HCC1143)
```

3 Filtering

Filtering windows is recommended to reduce the amount of false positives.
Once GC content is accounted for, the basic trimming filter discards those windows with extremely low or high read counts. The amount of windows to filter out is given in quantiles (from 0 to 1). For instance, a value of 0.01 discards 1 per cent of the windows with at least one read on both the upper and lower extremes. Two values can be given for the upper and lower quantiles, respectively. Withouth a matched normal, a value of 1 for the upper quantile automatically selects its threshold.
The mapping quality and mappability filters try to discard windows in which the amount of read segments might be lower than expected due to reasons such as the number of times the segments appear in the genome. The mappability filter requires the corresponding genome annotation (hg18 and hg19 annotations are provided in the package). While mappability ranges from 0 to 1, the mapping quality of reads ranges from 0 to 255. A mean mapping quality of 5 would indicate that the average probability of the reads being correctly mapped is 10e-5.
On samples sequenced with paired-end reads, the PEM-based filter discards windows in which the ratio of improper to proper reads is highest, where proper reads are those with correct orientation and whose insert size is within a reasonable limit. A correct orientation means that the reads in a pair are facing each other, whereas a sensible insert size depends on the library and is usually estimated by the aligner software.
The CNV-based filter, which requires the corresponding genome build, discards windows

where a common CNV spans most of their length. Annotation of common CNVs mapped to the hg18 and hg19 builds are provided in the package, where the CNVs are the ones detected by Altshuler et al. (**?**). Under most circumstances, this filter can be disregarded. In general, we suggest reviewing the function's output plots in order to adjust the filters to adequate values.
In this particular example, we apply the trimming and mapping quality filters, which do not require neither genome build annotation nor paired-end reads.

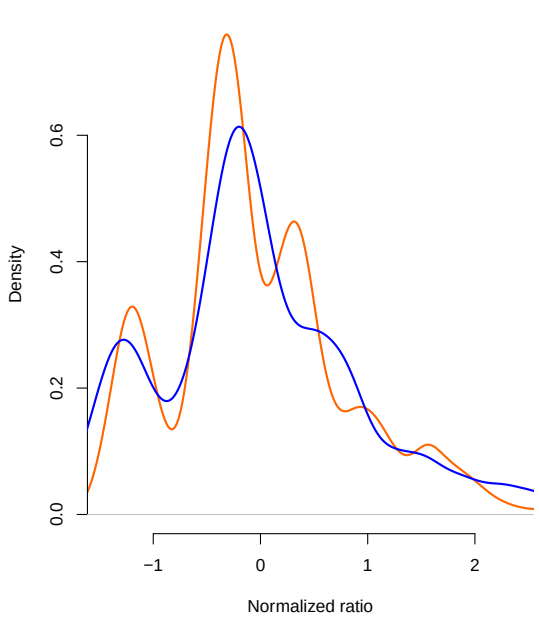
```
> rco = applyFilters(rco, trim.filter=1, mapq.filter=2)
```

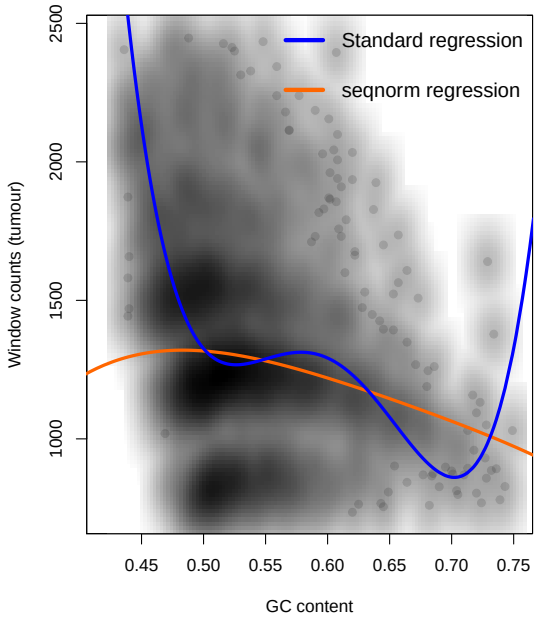


4 Normalization

GC content is the factor that most affects spurious read count variability. The method called seqnorm effectively corrects for this bias taking into account a possible linear relationship between copy number and GC content. If we had previously loaded a matched normal sample, a correction not against GC content but against the normal read count profile would have been made.
seqnorm uses GLAD (**?**) to segment a preliminar normalization and then performs regressions on each segment, which provides a robust estimate of the actual regression line.
Little parameterization is required internally by seqnorm. Only for the segmentation and for filtering segments. However, the same values can be used throughout samples, so the function takes no parameters.
The process can be sped up through parallelization, using as many cores as are available in the machine.

```
> rco = runSeqnorm(rco)
```





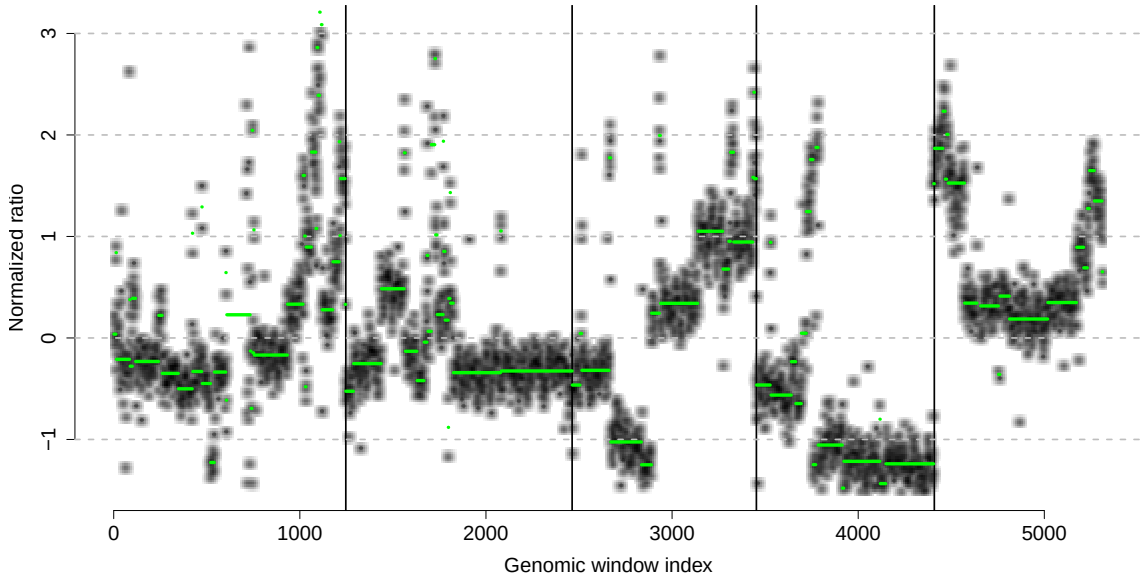
5 Segmentation and calling

The same segmentation method, GLAD (**?**), is used for the final segmentation. This step smooths the filtered and normalized profile, trying to keep only the significant changes. The level of smoothing can be controlled through a parameter if necessary (see `?runGLAD`).

```
> rco = runGLAD(rco)
```

Finally, the copy number calling is made. We found that visual assessment is necessary for copy number calling even after complex computational modelling. Hence, we offer the means to manually make the calling. Required input is: (i) the thresholds that separate the copy numbers in the profile and (ii) the copy number of those segmented values below the lowest indicated threshold.
We first plot the profile so that we can perform the visual assessment. Typically, the most common copy numbers are 2, 3 or 4 (i.e. mainly diploid, triploid or tetraploid profile).
In this example, we can observe that what seems to be the CN2 is below zero and CN1 a bit below -1, so we set the lowest threshold at -0.8 and set subsequent thresholds at intervals of 0.8. We also need to indicate which the lowest copy number is. In this case, CN1.

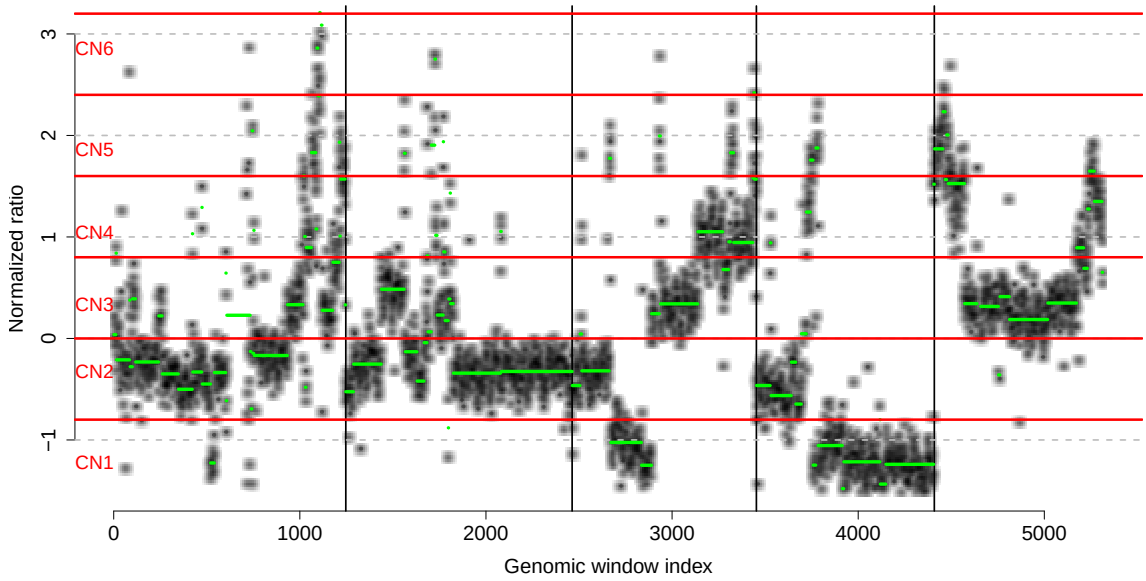
```
> plotCNProfile(rco)
```




```
> rco = applyThresholds(rco, seq(-0.8,4,by=0.8), 1)
```

Now that we have applied the thresholds, plotting the profile includes them in the plot of the finished analysis.

```
> plotCNProfile(rco)
```



At any step of the analysis, we can check a summary of the data and its processing step through the `summary` function.

```
> summary(rco)

Basic information:
  SeqCNAInfo object with 5314 200Kbp-long windows.
  PEM information is not available.
  Paired normal is not available.
  Genome and build unknown (chromosomes chr1 to chr5).
Total filtered windows: 212.
The profile is normalized and segmented.
Copy numbers called from 1 to 8.
  CN1:872 windows.
  CN2:2089 windows.
  CN3:1327 windows.
  CN4:579 windows.
  CN5:205 windows.
  CN6:17 windows.
  CN7:9 windows.
  CN8:4 windows.
```

A quick peek at the output reveals the profiles at the different analysis steps, where those windows that were initially filtered present NA values.

```
> head(rco@output)
```

```
  chrom win.start normalized segmented   CN
1  chr1         0         NA        NA <NA>
2  chr1       200         NA        NA <NA>
3  chr1       400         NA        NA <NA>
4  chr1       600     0.2170    0.0377    3
5  chr1       800     0.0122    0.0377    3
6  chr1      1000    -0.0623    0.0377    3
```

If we wanted to output the results to a file, we would use the `writeCNProfile` function.