Introduction to QDNAseq

Ilari Scheinin

April 27, 2020

Contents

1 Running QDNAseq

This is a short tutorial on how to use the *QDNAseq* package. It covers an example run using the included data set of chromosomes 7–10 of a low grade glioma (LGG) sample. First step is naturally to load the package.

```
> library(QDNAseq)
```

1.1 Bin annotations

Then we need to obtain bin annotations. These are available pre-calculated for genome build hg19 and bin sizes 1, 5, 10, 15, 30, 50, 100, 500, and 1000 kbp. They are available in the *QDNAseq.hg19* package, which has to be installed from Bioconductor separately. With that package installed, the bin annotations can be acquired as:

```
> bins <- getBinAnnotations(binSize=15)
Loaded bin annotations for genome 'hg19', bin size 15 kbp, and
```

```
experiment type 'SR50' from annotation package QDNAseq.hg19 v1.14.0
> bins
QDNAseq bin annotations for Hsapiens, build hg19.
Created by Ilari Scheinin with QDNAseq 0.7.5, 2014-02-06 12:48:04.
An object of class 'AnnotatedDataFrame'
  rowNames: 1:1-15000 1:15001-30000 ... Y:59370001-59373566 (206391
    total)
  varLabels: chromosome start ... use (9 total)
  varMetadata: labelDescription
```

If you are working with another genome build (or another species), see the section on generating the bin annotations.

1.2 Processing BAM files

Next step is to load the sequencing data from BAM files. This can be done for example with one of the commands below.

```
> readCounts <- binReadCounts(bins)
> # all files ending in .bam from the current working directory
>
> # or
>
> readCounts <- binReadCounts(bins, bamfiles="tumor.bam")
> # file 'tumor.bam' from the current working directory
>
> # or
>
> readCounts <- binReadCounts(bins, path="tumors")
> # all files ending in .bam from the subdirectory 'tumors'
```

This will return an object of class *QDNAseqReadCounts*. If the same BAM files will be used as input in future *R* sessions, option `cache=TRUE` can be used to cache intermediate files, which will speed up future analyses. Caching is done with package *R.cache*.

For large BAM files it is advisable to use the `chunkSize` parameter to control memory usage. A non-`NULL`, non-numeric value will use the length of the longest chromosome, effectively chunking by chromosome. A numeric value will use that many reads at a time. Note that total peak memory usage is controlled both by the chunk size and the number of parallel workers. See section 2.

For the purpose of this tutorial, we load an example data set of chromosomes 7–10 of low grade glioma sample LGG150.

```
> data(LGG150)
> readCounts <- LGG150
> readCounts

QDNAseqReadCounts (storageMode: lockedEnvironment)
assayData: 38819 features, 1 samples
  element names: counts
protocolData: none
phenoData
```

```
  sampleNames: LGG150
  varLabels: name reads used.reads
    expected.variance
  varMetadata: labelDescription
featureData
  featureNames: 7:1-15000
    7:15001-30000 ...
    10:135525001-135534747 (38819
    total)
  fvarLabels: chromosome start ...
    use (9 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

Plot a raw copy number profile (read counts across the genome), and highlight bins that will be removed with default filtering (Figure 1).

```
> plot(readCounts, logTransform=FALSE, ylim=c(-50, 200))

Plotting sample LGG150 (1 of 1) ...

> highlightFilters(readCounts, logTransform=FALSE,
+   residual=TRUE, blacklist=TRUE)

Highlighted 3,375 bins.
```

Apply filters and plot median read counts as a function of GC content and mappability (Figure 2). As the example data set only contains a subset of the chromosomes, the distribution looks slightly less smooth than expected for the entire genome.

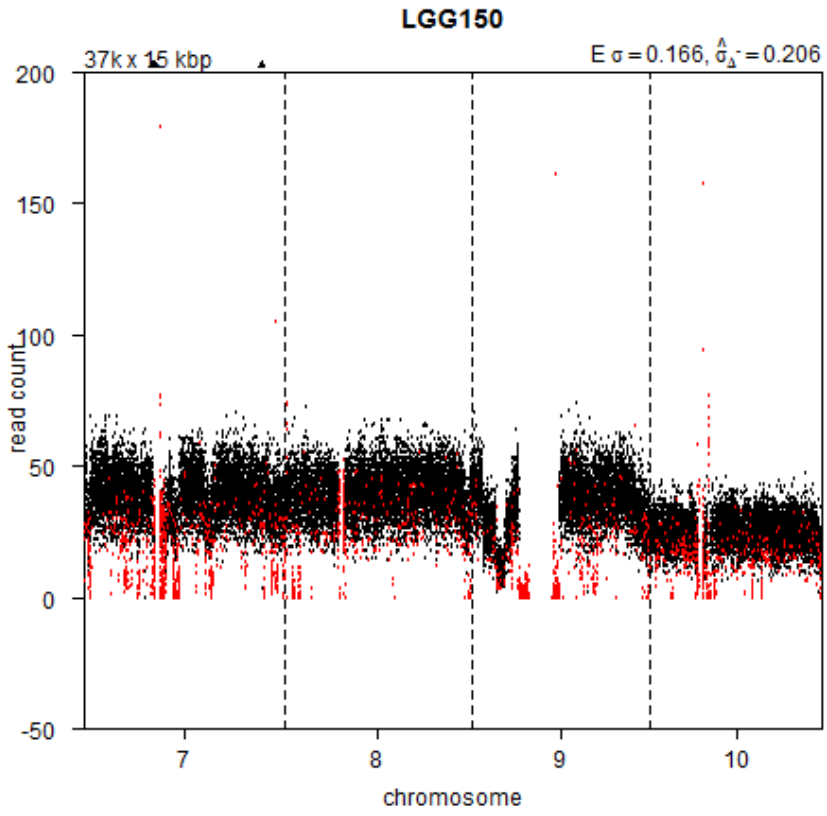
```
> readCountsFiltered <- applyFilters(readCounts, residual=TRUE, blacklist=TRUE)

38,819        total bins
38,819        of which in selected chromosomes
36,722        of which with reference sequence
33,347        final bins
```

```
> isobarPlot(readCountsFiltered)

Plotting sample LGG150 median read counts
```

Estimate the correction for GC content and mappability, and make a plot for the relationship between the observed standard deviation in the data and its read depth (Figure 3). The theoretical expectation is a linear relationship, which is shown in the plot with a black line. Samples with low-quality DNA will be noisier than expected and appear further above the line than good-quality samples.

```
> readCountsFiltered <- estimateCorrection(readCountsFiltered)

Calculating correction for GC content and mappability
    Calculating fit for sample LGG150 (1 of 1) ...
Done.
```

Figure 1: Read counts per bins
Highlighted with red are bins that will be filtered out.

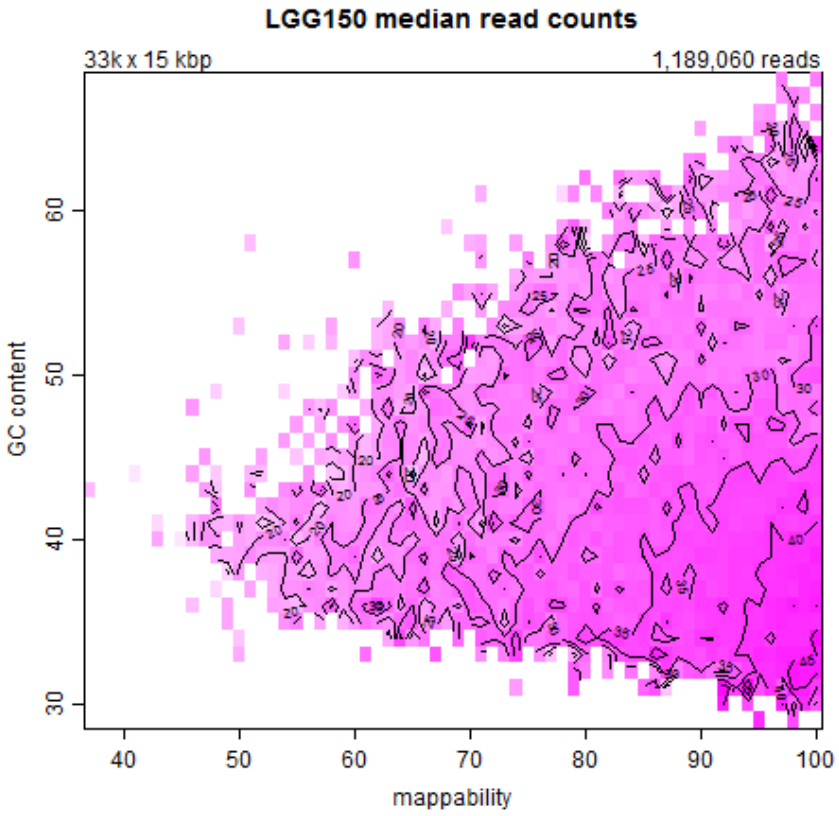



```
> noisePlot(readCountsFiltered)
```

Next, we apply the correction for GC content and mappability. This will return a *QDNAseqCopyNumbers* object, which we then normalize, smooth outliers, and plot the copy number profile (Figure 4).

```
> copyNumbers <- correctBins(readCountsFiltered)
> copyNumbers

QDNAseqCopyNumbers (storageMode: lockedEnvironment)
assayData: 38819 features, 1 samples
  element names: copynumber
protocolData: none
phenoData
  sampleNames: LGG150
  varLabels: name reads ...
    loess.family (6 total)
  varMetadata: labelDescription
featureData
```

Figure 2: Median read counts per bin shown as a function of GC content and mappability




```
    featureNames: 7:1-15000
      7:15001-30000 ...
      10:135525001-135534747 (38819
      total)
    fvarLabels: chromosome start ...
      use (9 total)
    fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

```
> copyNumbersNormalized <- normalizeBins(copyNumbers)
```

```
Applying median normalization ...
```

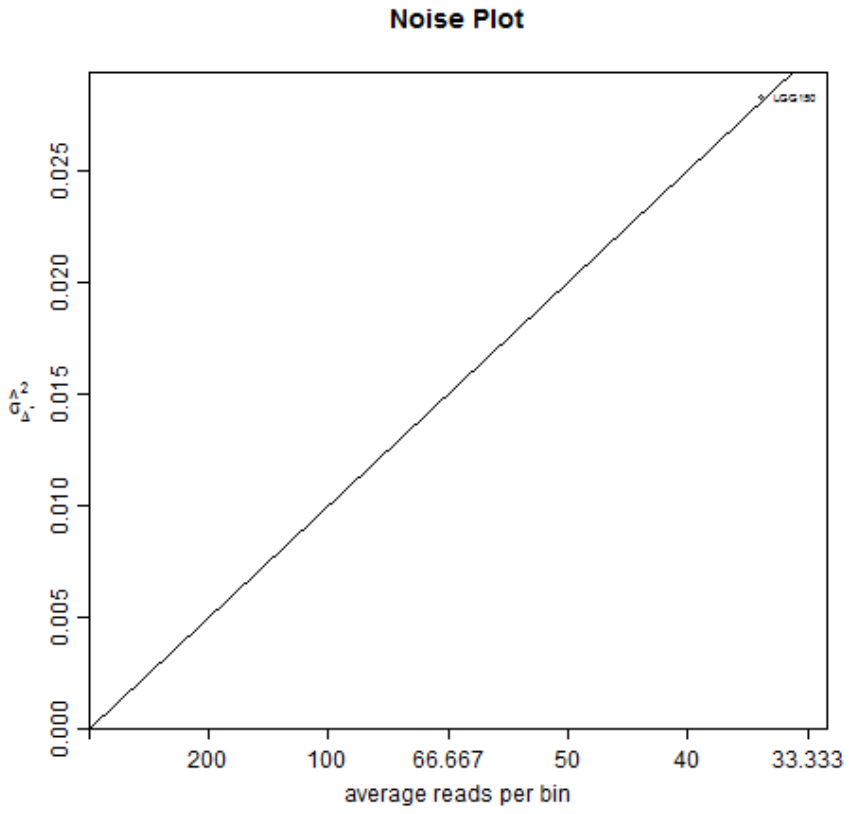
```
> copyNumbersSmooth <- smoothOutlierBins(copyNumbersNormalized)
```

```
Smoothing outliers ...
```

```
> plot(copyNumbersSmooth)
```

```
Plotting sample LGG150 (1 of 1) ...
```

Figure 3: The relationship between sequence depth and noise



Data is now ready to be analyzed with a downstream package of choice. For analysis with an external program or for visualizations in *IGV*, the data can be exported to a file.

```
> exportBins(copyNumbersSmooth, file="LGG150.txt")
> exportBins(copyNumbersSmooth, file="LGG150.igv", format="igv")
> exportBins(copyNumbersSmooth, file="LGG150.bed", format="bed")
```

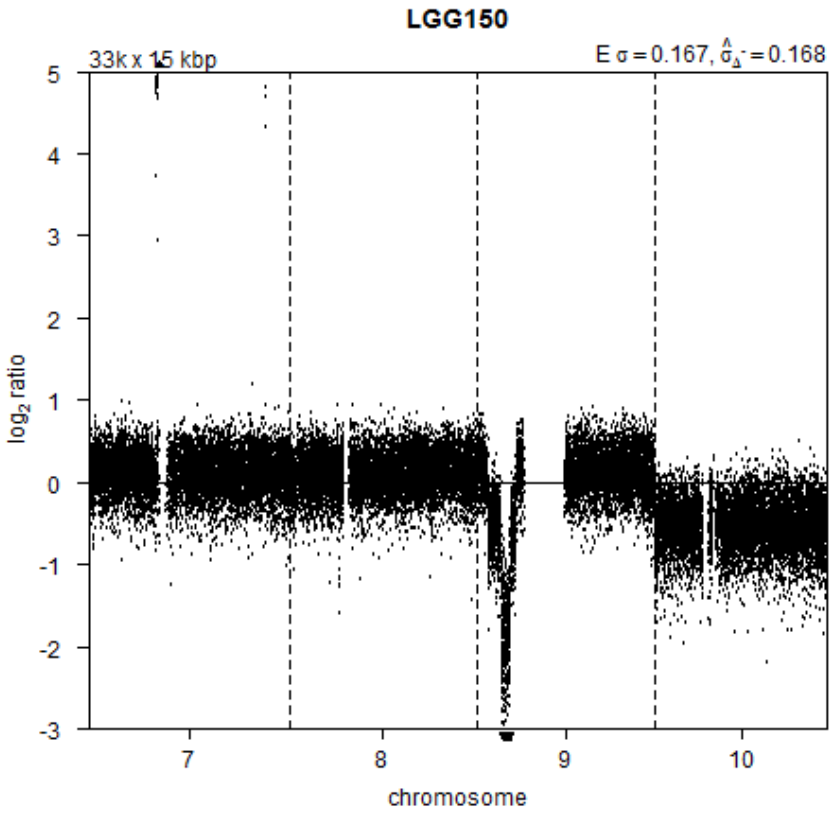
1.3 Downstream analyses

Segmentation with the *CBS* algorithm from *DNAcopy*, and calling copy number aberrations with *CGHcall* or cutoffs have been implemented for convenience.

By default, segmentation uses a $\log_2$-transformation, but a sqrt(x + 3/8) can also be used as it stabilizes the variance of a Poisson distribution (Anscombe transform):

```
> copyNumbersSegmented <- segmentBins(copyNumbersSmooth, transformFun="sqrt")

Performing segmentation:
    Segmenting: LGG150 (1 of 1) ...

> copyNumbersSegmented <- normalizeSegmentedBins(copyNumbersSegmented)
```

Figure 4: Copy number profile after correcting for GC content and mappability




```
> plot(copyNumbersSegmented)

Plotting sample LGG150 (1 of 1) ...
```

Tune segmentation parameters and iterate until satisfied. Next, call aberrations, and plot the final results.

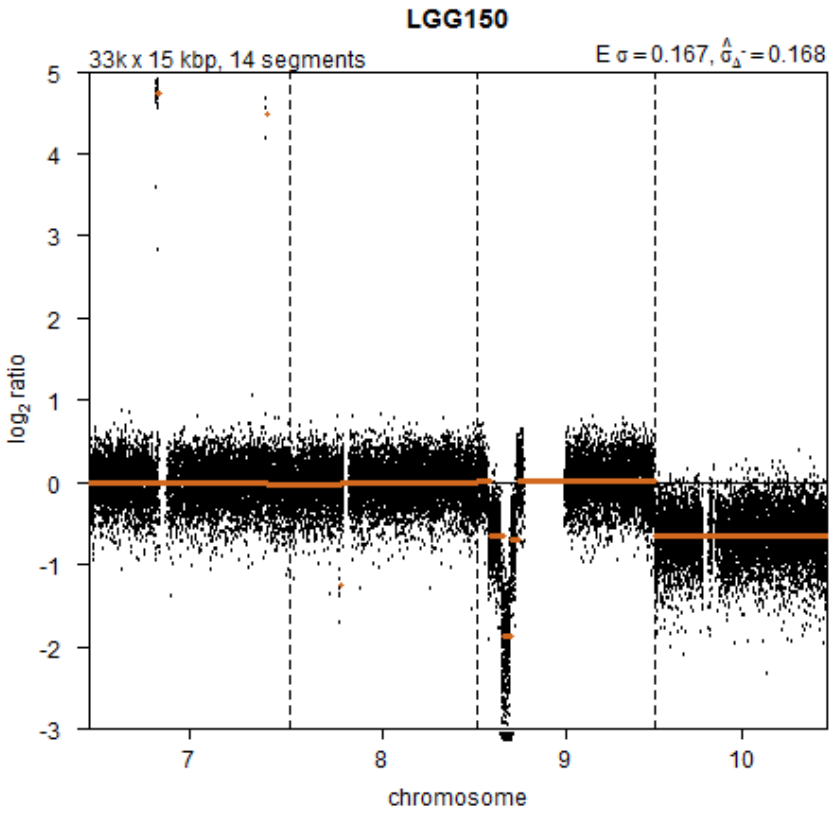
```
> copyNumbersCalled <- callBins(copyNumbersSegmented)

[1] "Total number of segments present in the data: 14"
[1] "Number of segments used for fitting the model: 11"
```

```
> plot(copyNumbersCalled)

Plotting sample LGG150 (1 of 1) ...
```

Called data can be exported as VCF file or SEG for further downstream analysis.

```
> exportBins(copyNumbersCalled, format="vcf")
> exportBins(copyNumbersCalled, format="seg")
```

Figure 5: Copy number profile after segmenting



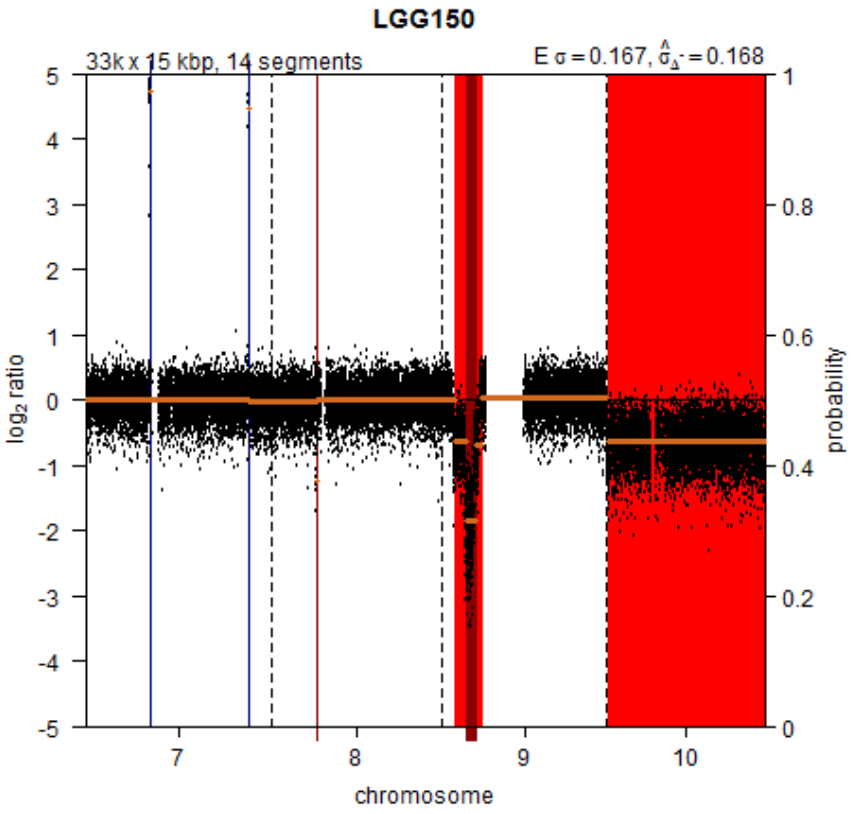
It should be noted that *CGHcall* (which `callBins()` uses by default) was developed for the analysis of sets of cancer samples. It is based on a mixture model, and when there are not enough aberrations present in the data, model fitting can fail. This can happen especially with non-cancer samples, and/or when analyzing individual cases instead of larger data sets.

If *CGHcall* fails, `callBins()` can also perform simple cutoff-based calling by setting parameter `method="cutoff"`. The default cutoff values are based on the assumption of uniform cell populations, and in case of cancer samples will most likely need calibration by adjusting parameter `cutoffs`.

Finally, for other downstream analyses, such as running *CGHregions*, it might be useful to convert to a *cghCall* object.

```
> cgh <- makeCgh(copyNumbersCalled)
> cgh

cghCall (storageMode: lockedEnvironment)
assayData: 33347 features, 1 samples
  element names: calls, copynumber, probamp, probdloss, probgain, probloss, probnorm, segmented
protocolData: none
phenoData
```

Figure 6: Copy number profile after calling gains and losses




```
  sampleNames: LGG150
  varLabels: name reads ...
    loess.family (6 total)
  varMetadata: labelDescription
featureData
  featureNames: 7:45001-60000
    7:60001-75000 ...
    10:135420001-135435000 (33347
    total)
  fvarLabels: Chromosome Start ...
    use (9 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

This command can also be used to generate *cghRaw* or *cghSeg* objects by running it before segmentation or calling.

2 Parallel computation

QDNAseq supports parallel computing via the *future* package. All that is required is to select an appropriate *plan*.

The instructions below apply to all of *QDNAseq*'s own functions that support parallel processing. At the moment these include `estimateCorrection()`, `segmentBins()`, `createBins()`, and `calculateBlacklist()`. `binReadCounts()` parallelizes by chromosome when `chunkSize` is used.

However, when argument `method="CGHcall"` (which is the default), function `callBins()` calls function `CGHcall()` from package *CGHcall*, which uses another mechanism for parallel computation. For that, the number of processes to use should be specified with argument `ncpus`, with something along the lines of:

```
> copyNumbers <- callBins(..., ncpus=4)
```

2.1 Non-parallel processing

The default is to use single-core processing via "sequential" futures. This can be set explicitly with:

```
> future::plan("sequential")
```

2.2 Parallel processing on the current machine

To process data in parallel using multiple processes on the current machine, use the following:

```
> future::plan("multiprocess")
```

After that, all functions that support parallel processing will automatically use it. The *future* framework attempts to play nice with the current compute environment. It will automatically respect environment variables and *R* options that are used to limit the number of parallel works. It will also respect environment variables such as number of cores assigned to job scripts in high-performance compute (HPC) clusters. If no such restrictions are set, the default is to use all cores available. To explicitly set, and override other settings, the number of parallel workers, use argument *workers*, e.g.

```
> future::plan("multiprocess", workers=4)
```

For more details, see the *future* documentation.

2.3 Parallel processing on an ad-hoc cluster

To process data using multiple *R* sessions running on different machines, use something along the lines of:

```
> cl <- future::makeClusterPSOCK(...)
> future::plan("cluster", cluster=cl)
```

See package *future* for more details.

3 Sex chromosomes

By default, *QDNAseq* ignores sex chromosomes. In order to include them in the analysis, function `applyFilters()` should be run with argument `chromosomes=NA` to include both X and Y, or `chromosomes="Y"` to include X only.

However, this will also affect which chromosomes are used when calculating the LOESS correction with `estimateCorrection()`. Unless the data set consists of only females, this could be undesirable. The solution is to first filter out the sex chromosomes, run `estimateCorrection()`, and then reverse the filtering of sex chromosomes:

```
> readCounts <- binReadCounts(getBinAnnotations(15))
> readCounts <- applyFilters(readCounts)
> readCounts <- estimateCorrection(readCounts)
> readCounts <- applyFilters(readCounts, chromosomes=NA)
> copyNumbers <- correctBins(readCounts)
```

Running `estimateCorrection()` and `correctBins()` with a different set of bins can have one side effect. This is caused by the fact that there can be bins in the sex chromosomes with a combination of GC content and mappability that is not found anywhere else in the genome. This will cause those bins to miss a correction estimate altogether, and these bins will be filtered out from subsequent steps by `correctBins()`. If this happens, it will print out a message specifying the number of bins affected.

Another possible approach is to allow extrapolation while calculating the LOESS correction. But please do note that the effect of extrapolation has not been properly evaluated.

```
> readCounts <- estimateCorrection(readCounts,
+   control=loess.control(surface="direct"))
```

4 Generating bin annotations

This section describes how bin annotations have been created for the hg19 build of the human reference genome, and can be applied for other genome builds and species. The first step is to create the bins based on chromosome sizes, and calculate their GC content and proportion of characterized nucleotides (non-N bases in the reference sequence). For this, the corresponding *BSgenome* package is needed.

```
> # load required packages for human reference genome build hg19
> library(QDNAseq)
> library(Biobase)
> library(BSgenome.Hsapiens.UCSC.hg19)
> # set the bin size
> binSize <- 15
> # create bins from the reference genome
> bins <- createBins(bsgenome=BSgenome.Hsapiens.UCSC.hg19, binSize=binSize)
```

The result is a *data.frame* with columns `chromosome`, `start`, `end`, `gc`, and `bases`. Next step is to calculate the average mappabilities, which requires a mappability file in the `bigWig` format and the *bigWigAverageOverBed* binary. The mappability file can be generated with *GEnomic Multi-Tool (GEM) Mapper* part of the *GEM library* from the reference genome sequence. Or it might be available directly, as was the case for hg19, and file '`wgEncodeCrgMapabilityAlign50mer.bigWig`' downloaded from ENCODE's download section of the UCSC Genome Browser. The *bigWigAverageOverBed* binary can also be downloaded from UCSC Genome Browser's Other utilities section.

```
> # calculate mappabilites per bin from ENCODE mapability tracks
> bins$mappability <- calculateMappability(bins,
+   bigWigFile="/path/to/wgEncodeCrgMapabilityAlign50mer.bigWig",
+   bigWigAverageOverBed="/path/to/bigWigAverageOverBed")
```

If there are genomic regions that should excluded from analyses, such as ENCODE's Blacklisted Regions, the percentage overlap between the generated bins and these regions can be calculated as follows. The regions to be excluded need to be in the `BED` format, like files '`wgEncodeDacMapabilityConsensusExcludable.bed`' and '`wgEncodeDukeMapabilityRegionsExcludable.bed`' that were downloaded from ENCODE's download section of the UCSC Genome Browser for hg19.

```
> # calculate overlap with ENCODE blacklisted regions
> bins$blacklist <- calculateBlacklist(bins,
+   bedFiles=c("/path/to/wgEncodeDacMapabilityConsensusExcludable.bed",
+              "/path/to/wgEncodeDukeMapabilityRegionsExcludable.bed"))
```

For any list of regions, the percentage of bin overlap can be calculated by using the following command.

```
> # generic calculation of overlap with blacklisted regions
> bins$blacklist <- calculateBlacklistByRegions(bins,
+   cbind(chromosome, bpStart, bpEnd))
```

To calculate median residuals of the LOESS fit from a control dataset, the following command can be used. For the pre-generated annotations, the control set used is 38 samples from the 1000 Genomes Project. See the next section on how those were downloaded.

```
> # load data for the 1000 Genomes (or similar) data set, and generate residuals
> ctrl <- binReadCounts(bins, path="/path/to/control-set/bam/files")
> ctrl <- applyFilters(ctrl, residual=FALSE, blacklist=FALSE,
+   mappability=FALSE, bases=FALSE)
> bins$residual <- iterateResiduals(ctrl)
```

The column `use` specifies whether each bin should be used for subsequent analyses by default. The command `applyFilters()` will change its value accordingly. By default, bins in the sex chromosomes, or with only uncharacterized nucleotides (N's) in their reference sequence, are flagged for exclusion.

```
> # by default, use all autosomal bins that have a reference sequence
> # (i.e. not only N's)
> bins$use <- bins$chromosome %in% as.character(1:22) & bins$bases > 0
```

Optionally, the resulting *data.frame* can be converted to an *AnnotateDataFrame* and metadata added for the columns.

```
> # convert to AnnotatedDataFrame and add metadata
> bins <- AnnotatedDataFrame(bins,
+   varMetadata=data.frame(labelDescription=c(
+   "Chromosome name",
+   "Base pair start position",
+   "Base pair end position",
+   "Percentage of non-N nucleotides (of full bin size)",
+   "Percentage of C and G nucleotides (of non-N nucleotides)",
+   "Average mappability of 50mers with a maximum of 2 mismatches",
+   "Percent overlap with ENCODE blacklisted regions",
+   "Median loess residual from 1000 Genomes (50mers)",
+   "Whether the bin should be used in subsequent analysis steps"),
+   row.names=colnames(bins)))
```

For the pre-generated annotations, some additional descriptive metadata has also been added.

```
> attr(bins, "QDNAseq") <- list(
+   author="Ilari Scheinin",
+   date=Sys.time(),
+   organism="Hsapiens",
+   build="hg19",
+   version=packageVersion("QDNAseq"),
+   md5=digest::digest(bins@data),
+   sessionInfo=sessionInfo())
```

5 Downloading 1000 Genomes samples

This section defines the criteria that were used to download samples from the 1000 Genomes Project for the pre-generated bin annotations.

```
> # download table of samples
> urlroot <- "ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp"
> g1k <- read.table(file.path(urlroot, "sequence.index"),
+   header=TRUE, sep="\t", as.is=TRUE, fill=TRUE)
> # keep cases that are Illumina, low coverage, single-read, and not withdrawn
> g1k <- g1k[g1k$INSTRUMENT_PLATFORM == "ILLUMINA", ]
> g1k <- g1k[g1k$ANALYSIS_GROUP == "low coverage", ]
> g1k <- g1k[g1k$LIBRARY_LAYOUT == "SINGLE", ]
> g1k <- g1k[g1k$WITHDRAWN == 0, ]
> # keep cases with read lengths of at least 50 bp
> g1k <- g1k[!g1k$BASE_COUNT %in% c("not available", ""), ]
> g1k$BASE_COUNT <- as.numeric(g1k$BASE_COUNT)
> g1k$READ_COUNT <- as.integer(g1k$READ_COUNT)
> g1k$readLength <- g1k$BASE_COUNT / g1k$READ_COUNT
> g1k <- g1k[g1k$readLength > 50, ]
> # keep samples with a minimum of one million reads
> readCountPerSample <- aggregate(g1k$READ_COUNT,
+   by=list(sample=g1k$SAMPLE_NAME), FUN=sum)
> g1k <- g1k[g1k$SAMPLE_NAME %in%
+   readCountPerSample$sample[readCountPerSample$x >= 1e6], ]
> g1k$fileName <- basename(g1k$FASTQ_FILE)
> # download FASTQ files
> for (i in rownames(g1k)) {
+   sourceFile <- file.path(urlroot, g1k[i, "FASTQ_FILE"])
+   destFile <- g1k[i, "fileName"]
+   if (!file.exists(destFile))
+     download.file(sourceFile, destFile, mode="wb")
+ }
```

Next, reads were trimmed to 50 bp, and the multiple files for each sample (as defined by column SAMPLE_NAME) were combined by concatenating the FASTQ files together. Finally, they were aligned with *BWA* allowing two mismatches and end-trimming of bases with qualities below 40 (options `-n 2 -q 40`).

6 Session information

The version number of *R* and packages loaded for generating the vignette were:

- R version 4.0.0 (2020-04-24), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`
- Random number generation:
- RNG: `L'Ecuyer-CMRG`
- Normal: `Inversion`
- Sample: `Rejection`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: QDNAseq 1.24.0
- Loaded via a namespace (and not attached): Biobase 2.48.0, BiocGenerics 0.34.0, BiocManager 1.30.10, BiocParallel 1.22.0, BiocStyle 2.16.0, Biostrings 2.56.0, CGHbase 1.48.0, CGHcall 2.50.0, DNAcopy 1.62.0, GenomeInfoDb 1.24.0, GenomeInfoDbData 1.2.3, GenomicRanges 1.40.0, IRanges 2.22.0, R.methodsS3 1.8.0, R.oo 1.23.0, R.utils 2.9.2, RCurl 1.98-1.2, Rcpp 1.0.4.6, Rsamtools 2.4.0, S4Vectors 0.26.0, XVector 0.28.0, bitops 1.0-6, codetools 0.2-16, compiler 4.0.0, crayon 1.3.4, digest 0.6.25, evaluate 0.14, future 1.17.0, future.apply 1.5.0, globals 0.12.5, htmltools 0.4.0, impute 1.62.0, knitr 1.28, limma 3.44.0, listenv 0.8.0, marray 1.66.0, matrixStats 0.56.0, parallel 4.0.0, rlang 0.4.5, rmarkdown 2.1, stats4 4.0.0, tools 4.0.0, xfun 0.13, yaml 2.2.1, zlibbioc 1.34.0