Introduction to ChIPseqR

Peter Humburg

January 23, 2026

1 Introduction

The `ChIPseqR` package was developed for the analysis of nucleosome ChIP-seq data. It is suitable for the high resolution analysis of end-sequenced nucleosomes after MNase digest and can be used for the analysis of nucleosome positioning as well as histone modification experiments. The parameters of the model used by `ChIPseqR` provide some flexibility and choosing them appropriately should allow for the analysis of other types of experiments although this has not been tested extensively. In the following sections we will discuss the application of `ChIPseqR` to simulated nucleosome sequencing data to demonstrate its basic functionality. This is followed by some details on the underlying method and a discussion of how this method may be adapted to other types of experiments.

2 Simulating data

We start by simulating a small dataset representing reads from a nucleosome positioning experiment. The information in this section details the data used for the following examples. To quickly see an application of `ChIPseqR` just skip ahead to the next section.

For the purpose of this example we will restrict ourselves to a very small genome and relatively few reads. This keeps time and memory requirements low but may affect `ChIPseqR`'s ability to locate binding sites. While it was designed to handle low coverage data it works better with genomes that are larger than the 2Mb we will simulate here. Also note that this simulation is somewhat simplistic and is used here for demonstration purposes only. If we were interested in carrying out a serious simulation study for ChIP-seq data we could use a more sophisticated approach[1] but this will be sufficient to demonstrate the basic use of `ChIPseqR`.

The first step in our simple simulation is to define some nucleosome positions. We will assume that the distance between the centres of adjacent nucleosomes lies between 170 and 300 bp and follows a negative binomial distribution with mean 200. In addition we will allow nucleosomes to be missing, introducing a gap of 400 bp between adjacent nucleosomes. Here we generate the distances between nucleosomes and translate them into nucleosome positions:

```
> set.seed(1)
> dist <- sample(c(170:300, 400), 10100, prob=c(dnbinom(0:130, mu=30, size=5), 0.2), replace
> pos <- cumsum(dist)
> pos <- pos[pos < 2e6 - 200]
```

Now we generate read positions relative to binding sites. For this purpose we will make a number of simplifying assumptions:

1. Reads fall within 15 bp of either end of a nucleosome (with forward strand reads preceding the binding site and reverse strand reads following it);
2. Reads may be located up to 5 bp inside the binding site;
3. Read positions within these 20 bp are uniformly distributed;

[1] The `ChIPsim` package available from Bioconductor provides functionality for this as does the simulation of Zhang *et al.* (2008) available at `www.gersteinlab.org/proj/chip-seq-simu`

4. The same number of forward and reverse strand reads are produced;
5. All nucleosomes cover exactly 147 bp;
6. Nucleosome positions are fixed;
7. Non-specific background reads are uniformly distributed throughout the genome.

Note that these assumptions are not used for the analysis and are not made by `ChIPseqR`, they just simplify the simulation.

We start by identifying all potential start positions for binding site related reads.

```
>         fwdRegion <- unlist(lapply(pos, function(x) (x-88):(x-68)))
>         revRegion <- unlist(lapply(pos, function(x) (x+68):(x+88)))
```

Then we sample 50,000 reads from each strand. This should give us an average of five reads per nucleosome and strand, not a lot of coverage.

```
>         fwd <- sample(fwdRegion, 5e4, replace=TRUE)
>         rev <- sample(revRegion, 5e4, replace=TRUE)
```

Finally we add 200,000 non-specific background reads to each strand. That is a lot of background noise so finding the nucleosomes will not be easy.

```
>         fwd <- c(fwd, sample(25:(2e6-25), 2e5, replace=TRUE))
>         rev <- c(rev, sample(25:(2e6-25), 2e5, replace=TRUE))
```

At this stage `fwd` and `rev` contain all positions of forward and reverse strand reads. The final step in our simulation is to organise the read position in a data frame that can then serve as input for `ChIPseqR`.

```
>         reads <- data.frame(chromosome="chr1", position=c(fwd, rev), length=25,
+                             strand=rep(c("+", "-"), each=250000))
```

3 Using `simpleNucCall`

```
>         library(ChIPseqR)
```

The easiest way to predict nucleosome positions with `ChIPseqR` is to call `simpleNucCall` with mapped reads. Here we will use the data frame created above but for real data it is usually more convenient to read the data into an `AlignedRead` object using the functionality provided by the `ShortRead` package. It is possible to simply pass the name of a file containing the mapped reads to `simpleNucCall` and they will be read in before the analysis. Here we first convert the read positions into read counts so that we can use the read counts again later. If we were not interested in inspecting the read counts later on we could just pass the data frame to `simpleNucCall` which would convert it into read counts for us.

```
>         counts <- strandPileup(reads, chrLen=2e6, extend=1, plot=FALSE)
>         nucs <- simpleNucCall(counts, bind=147, support=15, plot=FALSE)
```

Note the parameters in the above call. We specify the length of the binding and support regions as well as the chromosome length. The values chosen for binding and support region length do not match the values in the simulation perfectly (where they are 137 and 20 respectively). Below we will discuss briefly how these parameters can be estimated from the data. It is possible to obtain some diagnostic plots by setting `plot = TRUE`.

Diagnostic plots can also be obtained later by plotting the R objects created by `strandPileup` and `simpleNucCall`. For example, to genarate some diagnostic plots to assess how well the data fits the null distribution used by `ChIPseqR` we can simply plot `nucs`:

```
>         plot(nucs, type="density")
>         plot(nucs, type="qqplot")
```

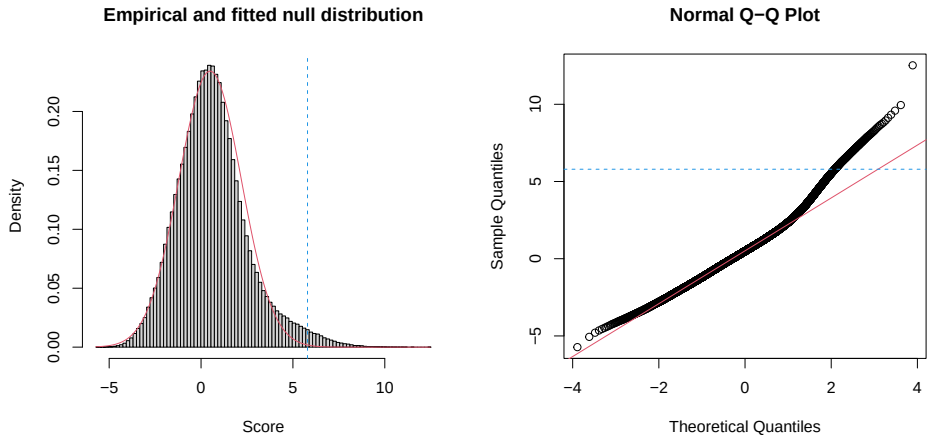

Figure 1: Diagnostic plots to assess fit of null distribution to observed binding site scores. The blue line marks the chosen significance cut-off.

The output is shown in Figure 1. The heavy right tail expected in the presence of nucleosomes in the sample is clearly visible. There is also some evidence that `ChIPseqR` overestimated the variance of the null distribution. As a result of this p-values computed from this fit will be conservative.

Here we have identified 4945 binding sites. Clearly some are missing but that is to be expected considering the relatively low coverage and high noise level. We can plot some of the data together with the binding site score and predicted nucleosome positions using `plotWindow`:

```
>         predicted <- peaks(nucs)[[1]][911]
>         plot(counts, chr="chr1", center=predicted, score=nucs, width=1000, type="window")
```

The output is shown in Figure 2. Note how we use `peaks` to obtain a list of predicted binding sites. To see how these predictions relate to the actual position of simulated nucleosomes we add some additional markers.

```
>         abline(v=pos[pos < predicted + 1000 & pos > predicted - 1000], col=3, lty=3)
```

As we can see in Figure 2 some nucleosome predictions overlap. These correspond to alternative positions of the same nucleosome. On real data this may indicate variations in the actual nucleosome position observed in the sample or simply uncertainty due to low coverage, high noise level or inaccurate choices of parameters. We can easily obtain non-overlapping predictions:

```
>         calls <- peaks(nucs)[[1]][c(1,which(diff(peaks(nucs)[[1]]) >= 170)+1)]
>         length(calls)
```

```
[1] 3571
```

Of course we would like to know how many of these are near actual nucleosome positions.

```
>         table(sapply(calls, function(x) any((x-20):(x+20) %in% pos)))
```

```
FALSE  TRUE
   30  3541
```

Of the 118 apparent false positives 76 are within 30 bp of a simulated nucleosome.

4 Some details on `callBindingSites`

Now that we have seen how to obtain binding site predictions it is time to take a closer look at how they are produced. Above we used `simpleNucCall` to identify nucleosome positions. The actual work required to locate binding sites is carried out by `callBindingSites`. The process of locating binding sites is divided into several stages:

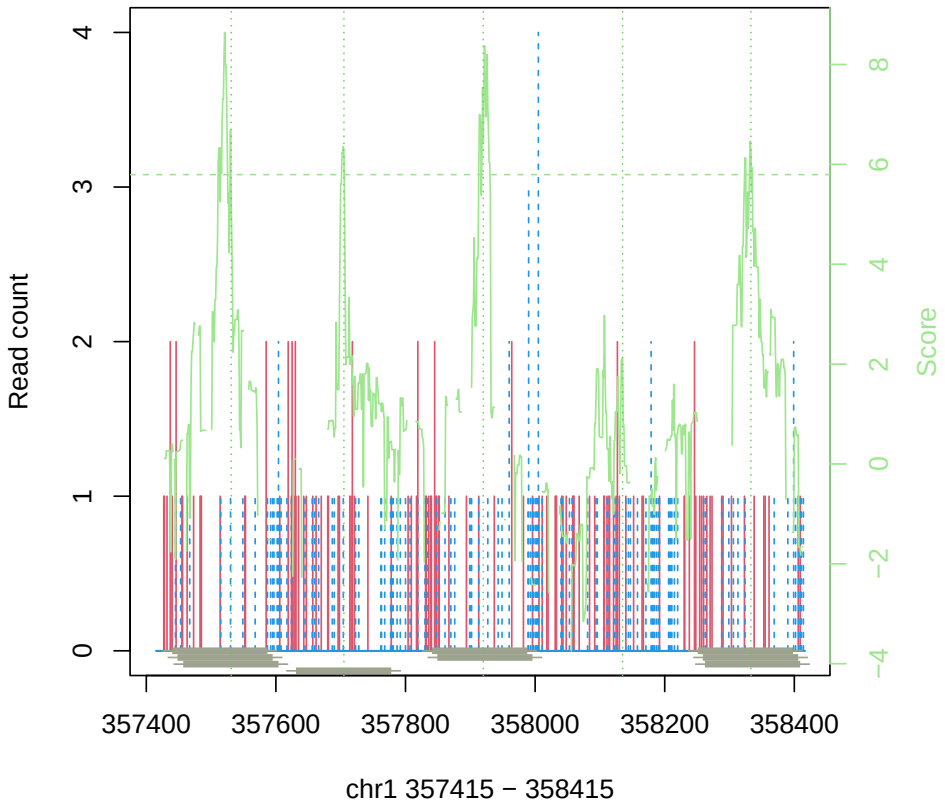

Figure 2: Read counts and predicted binding sites. Read counts on forward (red) and reverse (blue) strand are shown as vertical bars. The binding site score is shown in green with green boxes at the bottom indicating predicted nucleosome positions. The true position of simulated nucleosomes is indicated by vertical green lines.

1. Estimate length of binding and support region (if required);
2. Calculate binding site score;
3. Determine significance threshold;
4. Locate significant peaks in binding site score.

In this section we will take a closer look at these different stages to get a better understanding of how binding sites are identified and briefly introducing the functions used for each stage.

4.1 Estimating parameters

In practice it may not always be obvious what the best choice of parameters is. It is possible to get estimates for both parameters from the data by considering the cross-correlation between the read counts on the forward and reverse strand. The function `getBindLen` is provided for this purpose. However, it may not always produce good results. The approach taken in `getBindLen` is to identify the first two peaks in cross-correlation and to derive values for binding and support region length consistent with the location of these peaks. One of the main underlying assumptions is that the distribution of read counts in the support region is symmetric about the centre of the support region and the accuracy of results

depends largely on how well this assumption holds. It is also possible to supply one parameter and estimate the other which can improve results substantially if the supplied parameter value is accurate. It should also be noted that binding sites are assumed to occur in regular intervals and that they are located relatively close to each other. This may be approximately true for nucleosomes but does not hold for other proteins of interest.

Here is an example of how we might use `getBindLen` to estimate the parameters for our simulated data.

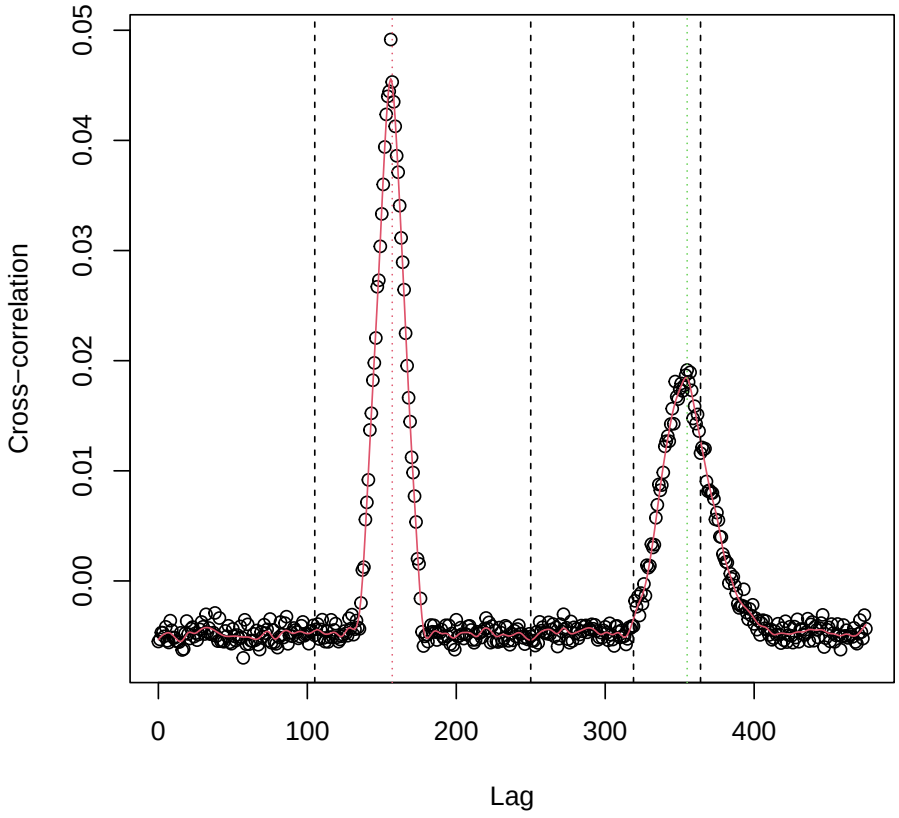

Figure 3: Cross-correlation between reads on forward and reverse strand used to estimate length of binding and support regions.

```
> bLen <- getBindLen(counts, bind = c(100,200), support = c(5, 50))
```

This tells `getBindLen` to look for peaks in the cross-correlation that correspond to a binding region length between 100 and 200 bp and a support region length between 5 and 50 bp. The resulting estimates are 75 bp for the binding region and 41 bp for the support regions. Since we know that the correct answer should be 137 and 20 bp respectively it is clear that this is not perfect.

To use this parameter estimation method as part of the binding site prediction we can simply pass `bind = c(100,200)` and `support = c(5, 50)` as arguments to `callBindingSites` (or `simpleNucCall`).

4.2 Scoring binding sites

During this step the read counts are scanned along the genome and each position is assigned a score indicating how likely it is to be the centre of a binding site. This involves a sliding window partitioned into three regions, the binding region in the centre and forward and reverse strand support regions on

either side. Read counts in each region are compared to a background estimate obtained from a larger window (the default is 2000 bp). A score is calculated for each region, with larger scores indication increasing departure from the background distribution, and these are then combined into a binding site score. The necessary computations are carried out by a call to `startScore`. Further parameters used here are `bgCutoff` and `supCutoff`. They can be used to limit the change in the estimate for adjacent background windows and between forward and reverse support regions. These parameters should be chosen between 0.5 and 1. Lower values correspond to a more restrictive cut-off. Setting `bgCutoff = 0.5` will essentially force the use of a uniform background (which is not recommended) while `bgCutoff = 1` allows arbitrarily large changes in background estimates. The cut-off for support regions works in a similar way. Generally lower cut-off values reduce the model's ability to adapt to changes in local coverage while larger values make it more sensitive to artificially large read counts observed in some locations.

4.3 Determining significance

Once binding site scores are calculated an attempt is made to assess their significance by estimating the parameters of the null distribution. To achieve this a (truncated) normal distribution is fitted to the left tail of the observed scores. This assumes that low scores are largely unaffected by the presence of binding sites. Computations carried out by `getCutoff` produce a binding site score cut-off corresponding to the requested false discovery rate and the parameters of the fitted null distribution.

4.4 Peak calling

During the final stage of the analysis significant peaks in the binding site score are identified and reported as predicted binding sites by `pickPeak`. This simply identifies all peaks that exceed the threshold determined in the previous step and identifies the location of the maximum of each peak as a binding site. Note that this may produce overlapping binding site predictions if two such peaks are located close to each other. By default peaks have to be seperated by at least one value below the threshold but in some cases, e.g. when peaks are relatively wide compared to the length of a binding site, we may want to relax this requirement such that peaks can be separated by a relatively low scoring region even if the score does not drop below the threshold. This can be achieved by using `sub = TRUE`.

5 Beyond nucleosomes

Although `ChIPseqR` has been designed to locate nucleosomes it should be possible to apply the same approach to other types of ChIP-seq experiments. The most obvious way to adjust `ChIPseqR` to different experiments is through the choice of binding and support region length. These relate to the physical length of the binding site and the length of DNA fragments. Generally a shorter binding site will require a shorter binding region but the best choice for this parameter also depends on the experimental protocol used. If DNA is sonicated it may be useful to further reduce the length of the binding region while increasing the length of support regions. Longer support regions will be required if the binding site is short compared to the fragment length.

6 Internal data representation

To reduce memory requirements the internal representation of read counts is compressed through run-length encoding. In some cases this can lead to an increase in required memory. Should this be the case for a particular dataset we can disable compression by calling `strandPileup` with `compress = FALSE`. If we just want to obtain an expanded representation of an already existing `ReadCounts` object we can use `decompress`:

```
>         counts2 <- decompress(counts)
```

In this case the decompressed representation is 2.5 times larger than the compressed one. For real datasets this factor can be substantially larger, especially when coverage is low.

A similar approach is used to compress the binding site score. Since the score is a floating point value a step function approximation is used to achieve better compression. This involves rounding the calculated score to a certain number of digits. The use of fewer digits will improve compression but also results in the loss of information and may affect results. The level of compression can be set by passing argument `digits` to `callBindingSite` or `simpleNucCall`. The default is 16 which results in low compression but retains the relevant information. The scoring method used by `ChIPseqR` is likely to produce missing values, especially in regions of low coverage. This can result in relatively long runs of `NA`'s that are always compressed. Again it is possible to expand the representation of binding site scores through a call to `decompress`. Note however that information that was lost during compression can only be restored through recalculating binding site scores at a lower compression setting. It is also possible to compress an expanded representation of read counts or binding site scores through a call to `compress`.

7 Session info

```
>         sessionInfo()

R version 4.5.2 (2025-10-31)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK version 3.12.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] ChIPseqR_1.65.0     S4Vectors_0.49.0    BiocGenerics_0.57.0
[4] generics_0.1.4

loaded via a namespace (and not attached):
 [1] Matrix_1.7-4                compiler_4.5.2
 [3] crayon_1.5.3                fBasics_4052.98
 [5] Rcpp_1.1.1                  ShortRead_1.69.2
 [7] SummarizedExperiment_1.41.0 Biobase_2.71.0
 [9] timsac_1.3.8-4              Rsamtools_2.27.0
[11] GenomicRanges_1.63.1        bitops_1.0-9
[13] Biostrings_2.79.4           GenomicAlignments_1.47.0
[15] parallel_4.5.2              png_0.1-8
[17] IRanges_2.45.0              Seqinfo_1.1.0
[19] timeSeries_4052.112         BiocParallel_1.45.0
[21] lattice_0.22-7              deldir_2.0-4
```

```
[23] XVector_0.51.0              S4Arrays_1.11.1
[25] latticeExtra_0.6-31         knitr_1.51
[27] DelayedArray_0.37.0         MatrixGenerics_1.23.0
[29] maketools_1.3.2             interp_1.1-6
[31] spatial_7.3-18              timeDate_4051.111
[33] RColorBrewer_1.1-3          HilbertVis_1.69.0
[35] rlang_1.1.7                 hwriter_1.3.2.1
[37] pwalign_1.7.0               xfun_0.56
[39] sys_3.4.3                   SparseArray_1.11.10
[41] cli_3.6.5                   grid_4.5.2
[43] evaluate_1.0.5              cigarillo_1.1.0
[45] codetools_0.2-20            buildtools_1.0.0
[47] abind_1.4-8                 jpeg_0.1-11
[49] matrixStats_1.5.0           tools_4.5.2
```