Contents

1 Introduction 2

2 Reading data 2

2.1 Reading Tiling Arrays 2

2.2 Importing BAM files 3

2.3 Next Generation Sequencing 4

2.4 MNase bias correction 5

3 Signal Smoothing and Nucleosome Calling 6

3.1 Noise removal 7

3.2 Peak detection and Nucleosome Calling 9

4 Exporting data 13

5 Generating synthetic maps 13

6 References 14

1 Introduction

The *nucleR2* package provides a high-level processing of genomic datasets focused in nucleosome positioning experiments, despite they should be also applicable to chromatin inmunoprecipitation (ChIP) experiments in general.

The aim of this package is not providing an all-in-one data analysis pipeline but complement those existing specialized libraries for low-level data importation and pre-processment into *R/Bioconductor* framework.

nucleR works with data from the two main high-troughput technologies available nowadays for ChIP: Next Generation Sequencing/NGS (ChIP-seq) and Tiling Microarrays (ChIP-on-Chip).

This is a brief summary of the main functions:

- Data import: `readBAM`, `processReads`, `processTilingArray`
- Data transformation: `coverage.rpm`, `filterFFT`, `controlCorrection`
- Nucleosome calling: `peakDetection`, `peakScoring`
- Visualization: `plotPeaks`
- Data generation: `syntheticNucMap`

For more details about the functions and how to use them refer to the *nucleR* manual.

This software was published in Bioinformatics Journal. See the paper for additional information (**???**).

2 Reading data

As mentioned previously, *nucleR* uses the pre-processed data of other lower level packages for data importation, supporting a few but common formats that should fulfill the requirements of most users.

`ExpressionSet` from package *Biobase* is used for Tiling Array experiments as described in *Starr* and other packages for the Tiling Array manipulation. This kind of experiments can be readed with the `processTilingArray` function.

`AlignedRead` from package *ShortRead* is recommended for NGS, covering most of the state of the art sequencing technologies. Additionally, support for reads in `RangedData` format is also provided (a range per read with a `strand` column).

2.1 Reading Tiling Arrays

Tiling Arrays are a cheap and fast way to have low-resolution nucleosome coverage maps. They have been widely used in literature (**???**), but complex statistical methods were needed for their processing (**???**).

This kind of microarrays cover a part of the genome with certain spacing between probes which causes a drop in the resolution and originates some problems. The nucleosome calling from Tiling Array data required hard work on bioinformatics side and use of heavy and artificious statistical machinery such as Hidden Markov Models (**???**) or higher order Bayesian Networks (**???**).

nucleR presents a new method based on a simple but effective peak calling method which achieves a great performance at low computing cost that will be presented in subsequent sections.

In order to standardize the data coming both from Tiling Arrays and NGS, the array fluorescence intensities (usually the ratio of the hybridization of nucleosomal and control sample) are converted to 1bp resolution by inferring the missed values from the neighboring probes. This is done by the function `processTilingArray`:

```
processTilingArray(data, exprName, chrPattern, inferLen=50)
```

An example of a processed dataset is provided in this package. See the help page of `tilingArray_preproc` for details on how it has been created. This object is a numeric vector covering the 8000 first positions of chromosome 1 in yeast (*Saccharomices cerevisiae* genome `SacCer1`).

```
library(nucleR)
library(ggplot2)
library(IRanges)
library(GenomicRanges)
data(nucleosome_tiling)
head(nucleosome_tiling, n=25)
#>  [1] 1.273222 1.281978 1.290734 1.299490 1.308246 1.352696 1.397145 1.441595
#>  [9] 1.486044 1.501795 1.517547 1.533298 1.549049 1.547577 1.546105 1.544633
#> [17] 1.543161 1.539886 1.536612 1.533337 1.530063 1.488922 1.447782 1.406642
#> [25] 1.365502
```

This values represent the normalized fluorescence intensity from hybridized sample of nucleosomal DNA versus naked DNA obtained from *Starr*. The values can be either direct observations (if a probe was starting at that position) or a inferred value from neighboring probes. This data can be passed directly to the filtering functions, as described later in the section 3.

2.2 Importing BAM files

Additionally, the function `importBAM`, allows to directly import into *R* the mapped reads of a NGS experiment contained in a *BAM* file. The user has to specify whether the file contains *paired-end* or *single-end* read fragments.

```
sample.file <- system.file("extdata", "cellCycleM_chrII_5000-25000.bam",
    package="nucleR")
reads <- readBAM(sample.file, type="paired")
head(reads)
#> GRanges object with 6 ranges and 0 metadata columns:
#>       seqnames    ranges strand
#>          <Rle> <IRanges>  <Rle>
#>   [1]    chrII 5790-5912      *
#>   [2]    chrII 5791-5920      *
#>   [3]    chrII 5809-5914      *
#>   [4]    chrII 5811-5983      *
#>   [5]    chrII 5815-5934      *
#>   [6]    chrII 5822-6096      *
```

```
#>   -------
#>   seqinfo: 17 sequences from an unspecified genome; no seqlengths
```

2.3 Next Generation Sequencing

NGS has become one of the most popular technique to map nucleosome in the genome in the last years (**???**). The drop of the costs of a genome wide sequencing together with the high resolution coverage maps obtained, made it the election of many scientists.

The package *ShortRead* allows reading of the data coming from many sources (Bowtie, MAQ, Illumina pipeline...) and has become one of the most popular packages in *R*/*Bioconductor* for NGS data manipulation.

A new *R* package, called *htSeqTools*, has been recently created to perform preprocessing and quality assesment on NGS experiments. *nucleR* supports most of the output generated by the functions on that package and recommends its use for quality control and correction of common biases that affect NGS.

nucleR handles `ShortRead` and `RangedData` data formats. The dataset `nucleosome_htseq` includes some NGS reads obtained from a nucleosome positioning experiment also from yeast genome, following a protocol similar to the one described in (**???**).

The paired-end reads coming from Illumina Genome Analyzer II sequencer were mapped using Bowtie and imported into *R* using *ShortRead*. Paired ends where merged and sorted according the start position. Those in the first 8000bp of chromosome 1 where saved for this example. Further details are in the reference (**???**):

```
data(nucleosome_htseq)
class(nucleosome_htseq)
#> [1] "GRanges"
#> attr(,"package")
#> [1] "GenomicRanges"
nucleosome_htseq
#> GRanges object with 18001 ranges and 0 metadata columns:
#>           seqnames    ranges strand
#>              <Rle> <IRanges>  <Rle>
#>       [1]     chr1     1-284      +
#>       [2]     chr1     5-205      +
#>       [3]     chr1     5-205      +
#>       [4]     chr1     5-209      +
#>       [5]     chr1     5-283      +
#>       ...      ...       ...    ...
#>   [17997]     chr1 7994-8151      +
#>   [17998]     chr1 7994-8151      +
#>   [17999]     chr1 7994-8151      +
#>   [18000]     chr1 7994-8152      +
#>   [18001]     chr1 7994-8152      +
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Now we will transform the reads to a normalized format. Moreover, as the data is paired-ended and we are only interested in mononucleosomes (which are typically 147bp), we will discard the reads with a length greater than 200bp, allowing margin for some underdigestion but discarding extra long reads. Note that the behaviour of `fragmentLen` is different for single-ended data, see the manual page of this function for detailed information.

As our final objective is identifying the nucleosome positions, and *Bioconductor*nucleR does it from the dyad, we will increase the sharpness of the dyads by removing some bases from the ends of each read. In the next example, we will create two new objects, one with the original paired-end reads and another one with the reads trimmed to the middle 40bp around the dyad (using the `trim` argument).

```
# Process the paired end reads, but discard those with length > 200
reads_pair <- processReads(nucleosome_htseq, type="paired", fragmentLen=200)

# Process the reads, but now trim each read to 40bp around the dyad
reads_trim <- processReads(nucleosome_htseq, type="paired", fragmentLen=200,
    trim=40)
```

The next step is obtain the coverage (the count of how many reads are in each position). The standard *IRanges* package function `coverage` will work well here, but it is a common practice to normalize the coverage values according to the total number of short reads obtained in the NGS experiment. The common used unit is *reads per milon* (r.p.m.) which is the coverage value divided by the total number of reads and multiplied per one milion. A quick and efficient way to do this with *nucleR* is the `coverage.rpm` function.[1]

```
# Calculate the coverage, directly in reads per million (r.p.m)
cover_pair <- coverage.rpm(reads_pair)
cover_trim <- coverage.rpm(reads_trim)
```

In Figure 1 we can observe the effect of `trim` attribute plotting both coverages. Note that the coverages are normalized in the range 0–1:

[1]Note that conversion in the example dataset gives huge values. This is because r.p.m. expects a large number of reads, and this dataset is only a fraction of a whole one. Also take into account that reads from single-ended (or trimmed reads) and reads from paired-ended could have different mean value of coverage

2.4 MNase bias correction

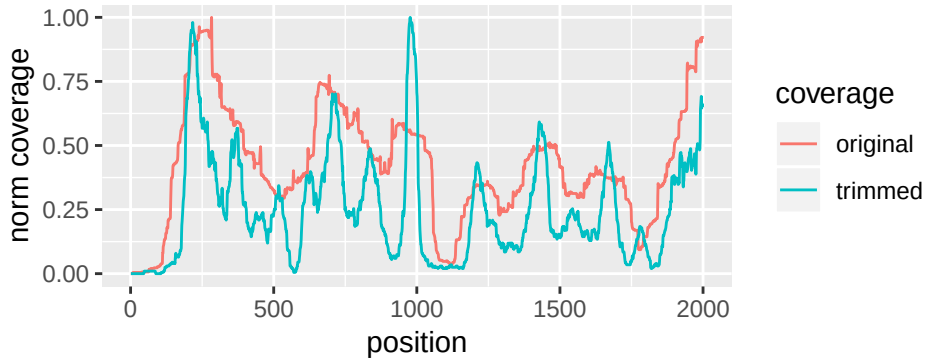

Figure 1: Variation in the sharpness of the peaks using `trim` attribute

The Microccocal Nuclease is a widely used enzyme that has been proved to have a biase for certain dinucleotide steps (**???**). In this package we offer a quick way to inspect the effect of such artifact by correcting the profiles of nucleosomal DNA reads with a mock sample of naked DNA digested with MNase.

The use of this function requires a paired-end control sample and a paired end or extended single-read nucleosomal DNA sample. A toy example generated using synthetic data can be found in Figure 2.

```
# Toy example
map <- syntheticNucMap(as.ratio=TRUE, wp.num=50, fuz.num=25)
exp <- coverage(map$syn.reads)
ctr <- coverage(map$ctr.reads)
corrected <- controlCorrection(exp, ctr)
```

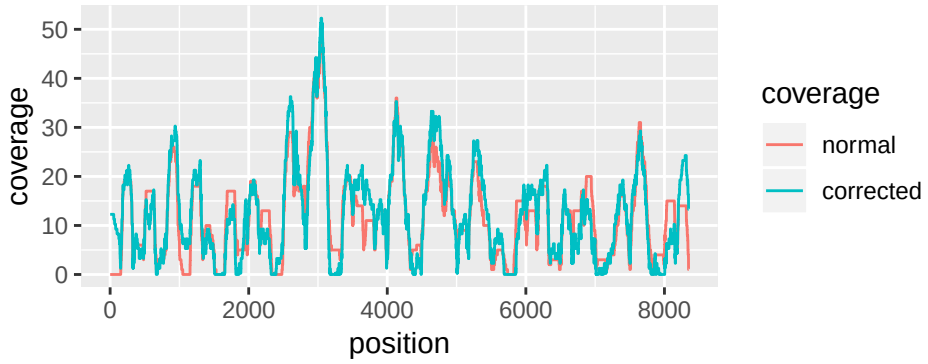

Figure 2: Toy example of MNase biase correction
Random nucleosomal and control reads have been generated using `synteticNucMap` function and corrected using `controlCorrect`.

3 Signal Smoothing and Nucleosome Calling

In the previous sections we converted the experimental data from NGS or Tiling Arrays to a continous, 1bp resolution signal. In this section we will remove the noise present in the data and score the peaks identified, giving place to the nucleosome calls.

Previously, in the literature, Hidden Markov Models, Support Vector Machines or other complex intelligent agents where used for this task (**???**). This was needed for dealing with the noise and uncertain characterization of the fuzzy positioning of the nucleosomes.

Despite this approach is a valid way to face the problem, the use of such artificious constructs is difficult to implement and sometimes requires a subjective modeling of the solution, constraining or at least conditioning the results observed.

The method presented here proposes to *keep it simple*, allowing the researcher to study the results he or she is interested *a posteriori*.

nucleR aim is to evaluate where the nucleosomes are located and how accurate that position is. We can find a nucleosome read in virtually any place in the genome, but some positions will show a high concentration and will allow us to mark this nucleosome as **well-positioned** whereas other will be less phased giving place to **fuzzy** or **de-localized** nucleosomes (**???**).

We think it's better to provide a detailed but convenient identification of the relevant nucleosome regions and score them according to its grade of fuzziness. From our point of view, every researcher should make the final decision regarding filtering, merging or classifying the nucleosomes according its necessities, and *nucleR* is only a tool to help in this *dirty* part of the research.

3.1 Noise removal

NGS and specially Tiling Array data show a very noisy profile which complicates the process of the nucleosome detection from peaks in the signal. A common approach used in the literature is smooth the signal with a sliding window average and then use a Hidden Markov Model to calculate the probabilities of having one or another state.

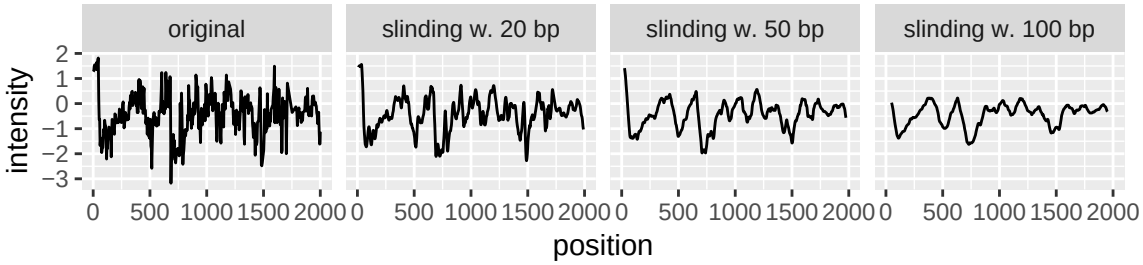

Figure 3: Original intensities from tiling array experiment
Smoothing using a sliding window of variable length (0, 20, 50 and 100 bp) is presented.

As can be seen in Figure 3, data needs some smoothing to be interpretable, but a simple sliding window average is not sufficient. Short windows allow too much noise but larger ones change the position and the shape of the peaks.

nucleR proposes a method of filtering based on the Fourier Analysis of the signal and the selection of its principal components.

Any signal can be described as a function of individual periodic waves with different frequencies and the combination of them creates more complex signals. The noise in a signal can be described as a small, non periodic fluctuations, and can be easily identified and removed (**???**).

nucleR uses this theory to transform the input data into the Fourier space using the Fast Fourier Transform (FFT). A FFT has a real and a imaginary component. The representation of the real component it's called the power spectrum of the signals and shows which are the frequencies that have more weight (power) in the signal. The low frequency components (so, very periodic) usually have a huge influence in the composite signal, but its rellevance drops as the frequency increases.

We can look at the power spectrum of the example dataset with the following command:

```
fft_ta <- filterFFT(nucleosome_tiling, pcKeepComp=0.01, showPowerSpec=TRUE)
```

In the Figure 4 only the half of the components are plotted, as the spectrum is repeated symmetrically respect to its middle point. The first component (not shown in the plot), has period 1, and, in practice, is a count of the lenght of the signal, so it has a large value.

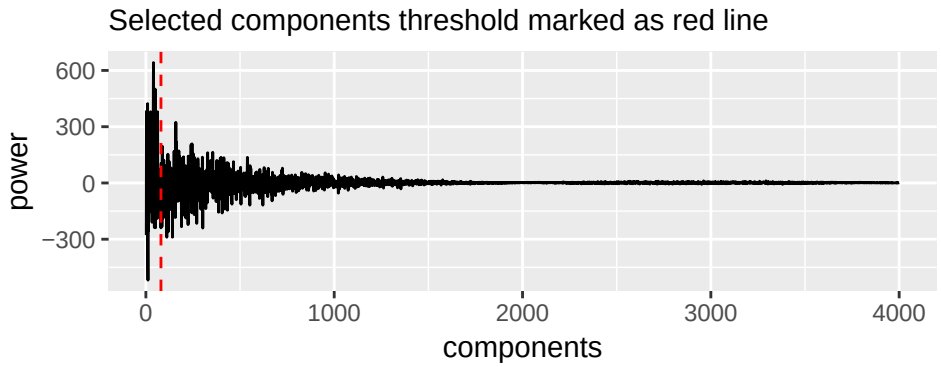

Figure 4: Power spectrum of the example Tiling Array data, percentile 1 marked with a dashed line

High frequency signals are usually echoes (repeating waves) of lower frequencies, i.e. a peak at 10 will be the sum of the pure frequence 10 plus the echo of the frequency 5 in its 2nd repetition. Echoes can be ignored without losing relevant information.

The approach *nucleR* follows is supposing that with just a small percentage of the components of the signal, the input signal can be recreated with a high precision, but without a significant amount of noise. We check empirically that with 1% or 2% of the components (this means account 1 or 2 components for each 100 positions of the genomic data) it's enough to recreate the signal with a very high correlation (>0.99). Tiling Array could require more smoothing (about 1% should be fine) and NGS has less noise and more components can be selected for fitting better the data (about 2%), See Figure 4 for the selected components in the example.

In order to easy the choice of the `pcKeepComp` parameter, *nucleR* includes a function for automatic detection of a fitted value that provides a correlation between the original and the filtered profiles close to the one specified. See the manual page of `pcKeepCompDetect` for detailed information.

In short, the cleaning process consists on converting the coverage/intensity values to the Fourier space, and knock-out (set to 0) the components greater than the given percentile in order to remove the noise from the profile. Then the inverse Fast Fourier Transform is applyied to recreate the filtered signal. In Figure 5 the filtered signal is overlapped to the raw signal.

The cleaning of the input has almost no effect on the position and shape of the peaks, mantaining a high correlation with the original signal but allowing achieve a great performance with a simple peak detection algorithm:

```
tiling_raw <- nucleosome_tiling
tiling_fft <- filterFFT(tiling_raw, pcKeepComp=0.01)
htseq_raw <- as.vector(cover_trim[[1]])
htseq_fft <- filterFFT(htseq_raw, pcKeepComp=0.02)

cor(tiling_raw, tiling_fft, use="complete.obs")
#> [1] 0.7153782
cor(htseq_raw, htseq_fft, use="complete.obs")
#> [1] 0.9937643
```

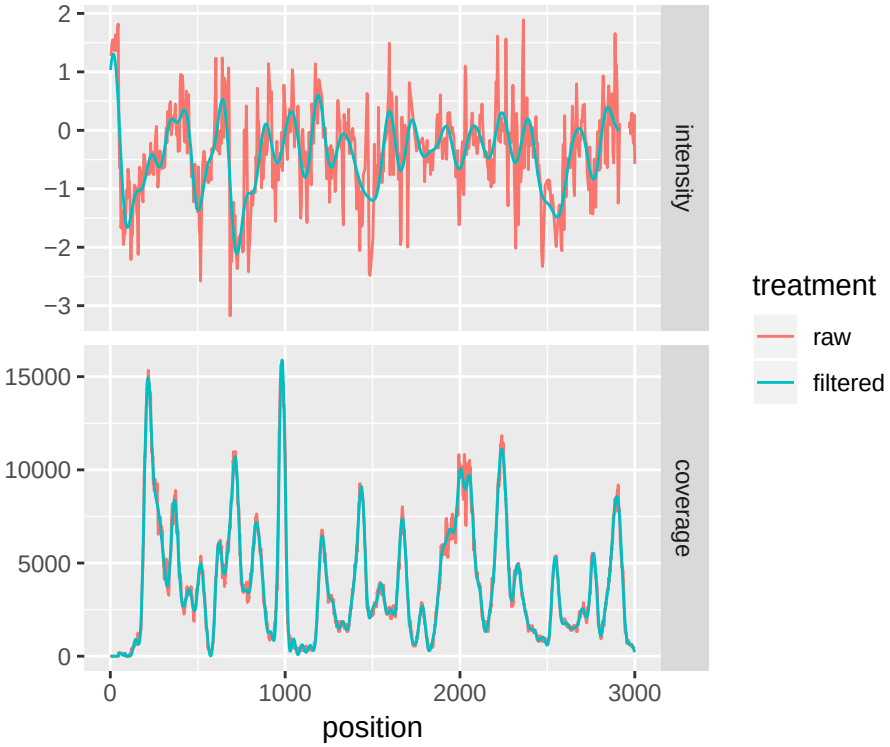

Figure 5: Filtering in Tiling Array (up, blue) (1% comp.) and NGS (down red) (2% comp.)

3.2 Peak detection and Nucleosome Calling

After noise removal, the calling for nucleosomes is easy to perform. In nucleosome positioning, in contrast with other similar experiments like ChIP, the problem for the peaks detection algorithms is deal with the presence of an irregular signal which causes lots of local maxima (i.e., peaks due to noise inside a real peak). Here, we avoid this problem applying the FFT filter, allowing the detection of peaks in a simple but efficient way just looking for changes in the trend of the profile. This is implemented in the `peakDetection` function and results can be represented with the function `plotPeaks`:

```
peaks <- peakDetection(htseq_fft, threshold="25%", score=FALSE)
peaks
#>  [1]  218  368  452  518  623  715  776  836  983 1213 1331 1437 1549 1603
#> [15] 1672 1785 1945 2005 2053 2241 2330 2546 2605 2702 2764 2899 3124 3285
#> [29] 3354 3518 3710 3869 4009 4137 4233 4305 4383 4524 4596 4832 4912 4981
#> [43] 5171 5241 5291 5366 5458 5529 5598 5688 5752 5906 5978 6065 6123 6238
#> [57] 6312 6410 6472 6669 6834 6892 7002 7048 7100 7152 7314 7405 7457 7545
#> [71] 7615 7684 7775 7932 8042

plotPeaks(peaks, htseq_fft, threshold="25%", ylab="coverage")
```

All the peaks above a threshold value are identified. Threshold can be set to 0 for detecting all the peaks, but this is not recommended as usually small fluctuations can apear in bottom part of the profile. This package also provides an automatic scoring of the peaks, which accounts for the two main features we are interested in: the height and the sharpness of the peak.

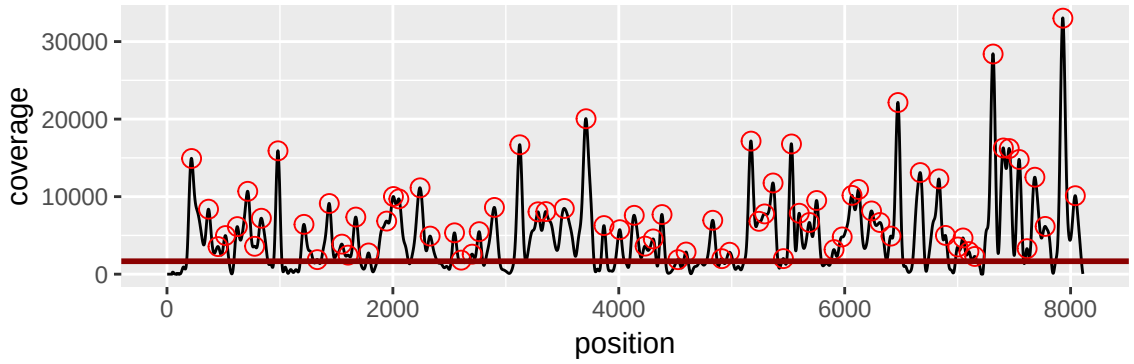

Figure 6: Output of `plotPeaks` function
Peaks are spotted in red and detection threshold marked with an horitzontal line.

The *height* of a peak is a direct measure of the reads coverage in the peak position, but represented as a probability inside a Normal distribution.

The *sharpness* is a measure of how fuzzy is a nucleosome. If a peak is very narrow and the surrounding regions are depleted, this is an indicator of a good positioned nucleosome, while wide peaks or peaks very close to each other are probably fuzzy nucleosomes (despite the coverage can be very high in this region).

Scores can be calculated with the `peakScoring` function or directly with the argument `score=TRUE` in `peakDetection`.

```
peaks <- peakDetection(htseq_fft, threshold="25%", score=TRUE)
head(peaks)
#>   peak     score
#> 1  218 0.9749612
#> 2  368 0.6824909
#> 3  452 0.2675856
#> 4  518 0.3829457
#> 5  623 0.4858367
#> 6  715 0.8408881

plotPeaks(peaks, htseq_fft, threshold="25%")
```

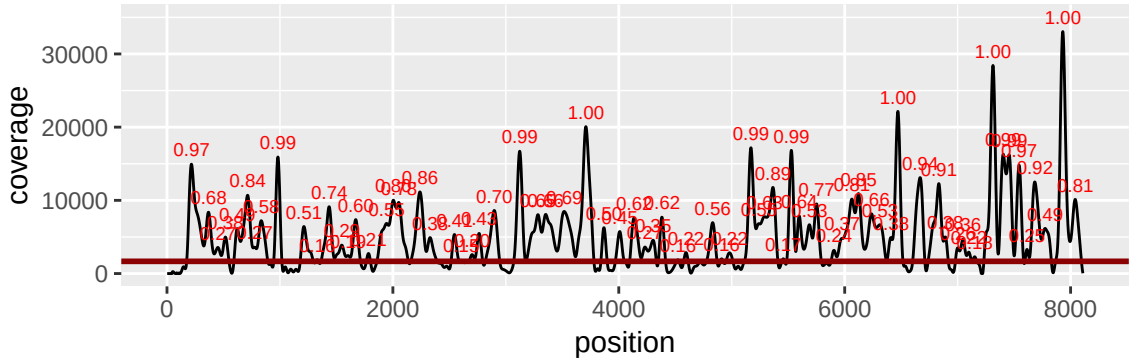

Figure 7: `plotPeaks` function with `score=TRUE`

The scores in Figure 7 only account for the punctual height of the peak. As said previously, this measure can be improved by accounting the fuzzyness of a nucleosome call (the sharpness of the peak). This requires a way to account for longer range peaks, which can be obtained with the `width` argument. In this way one can convert the identified nucleosome dyads to whole nucleosome length ranges and account for its degree of fuzzyness:

```
peaks <- peakDetection(htseq_fft, threshold="25%", score=TRUE, width=140)
peaks
#> GRanges object with 74 ranges and 3 metadata columns:
#>        seqnames    ranges strand |             score           score_w
#>           <Rle> <IRanges>  <Rle> |         <numeric>         <numeric>
#>    [1]        *   148-287      * | 0.821329600073532  0.66769801554711
#>    [2]        *   298-437      * |  0.63851684558774  0.59454274280948
#>    [3]        *   382-521      * | 0.315750617962796 0.363915608187116
#>    [4]        *   448-587      * | 0.518273290379434 0.653600845587843
#>    [5]        *   553-692      * | 0.518243691602914 0.550650655898527
#>    ...      ...       ...    ... .               ...               ...
#>   [70]        * 7475-7614      * | 0.864564673381752 0.755883863803246
#>   [71]        * 7545-7684      * |  0.21205553046579 0.176273435846357
#>   [72]        * 7614-7753      * | 0.787492153090483 0.654097601194428
#>   [73]        * 7705-7844      * | 0.471687940296344 0.455279255076558
#>   [74]        * 7862-8001      * | 0.913529783057904 0.827059566759111
#>                  score_h
#>                <numeric>
#>    [1] 0.974961184599953
#>    [2]    0.682490948366
#>    [3] 0.267585627738477
#>    [4] 0.382945735171026
#>    [5] 0.485836727307301
#>    ...               ...
#>   [70] 0.973245482960258
#>   [71] 0.247837625085224
#>   [72] 0.920886704986538
#>   [73]  0.48809662551613
#>   [74] 0.999999999356697
#>   -------
#>   seqinfo: 1 sequence from an unspecified genome; no seqlengths

plotPeaks(peaks, htseq_fft, threshold="25%")
```

Note than in Figure 8 overlapped peaks in a width and tall region are penalized, meanwhile the peaks with surrounding depleted regions have a higher relative score. This is the approach recommended for working with nucleosome calls.

Nucleosome calls filtering, merging or classification can be performed with standard Biocpkg{IRanges} functions, shuch as `reduce`, `findOverlaps` or `disjoint`.

The next example shows a simple way to merge those nucleosomes which are overlap accounting them as a fuzzy regions:

```
nuc_calls <- ranges(peaks[peaks$score > 0.1, ])
red_calls <- reduce(nuc_calls)
red_class <- RangedData(red_calls, isFuzzy=width(red_calls) > 140)
```

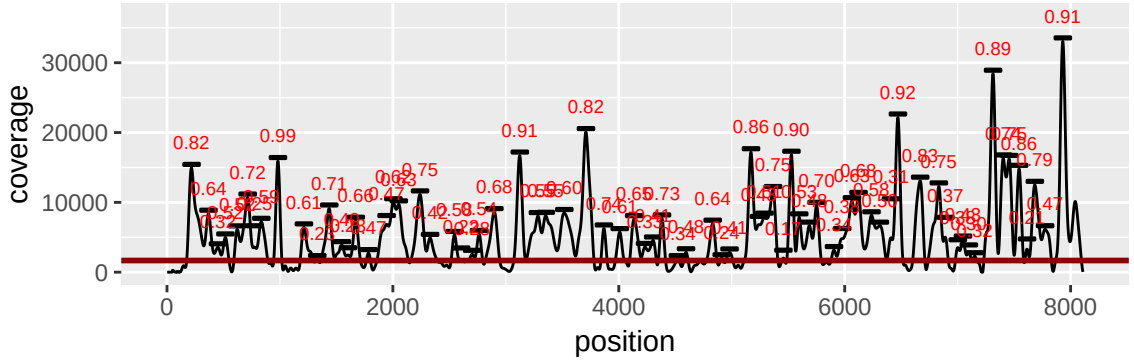

Figure 8: `plotPeaks` output with `score=TRUE` and `width=140`

```
red_class
#> RangedData with 20 rows and 1 value column across 1 space
#>        space    ranges   |   isFuzzy
#>     <factor> <IRanges>   | <logical>
#> 1          1   148-287   |     FALSE
#> 2          1   298-905   |      TRUE
#> 3          1  913-1052   |     FALSE
#> 4          1 1143-1854   |      TRUE
#> 5          1 1875-2122   |      TRUE
#> 6          1 2171-2399   |      TRUE
#> 7          1 2476-2968   |      TRUE
#> 8          1 3054-3193   |     FALSE
#> 9          1 3215-3423   |      TRUE
#> ...      ...       ... ...       ...
#> 12         1 3799-4452   |      TRUE
#> 13         1 4454-4665   |      TRUE
#> 14         1 4762-5050   |      TRUE
#> 15         1 5101-5821   |      TRUE
#> 16         1 5836-6541   |      TRUE
#> 17         1 6599-6738   |     FALSE
#> 18         1 6764-7221   |      TRUE
#> 19         1 7244-7844   |      TRUE
#> 20         1 7862-8001   |     FALSE
```

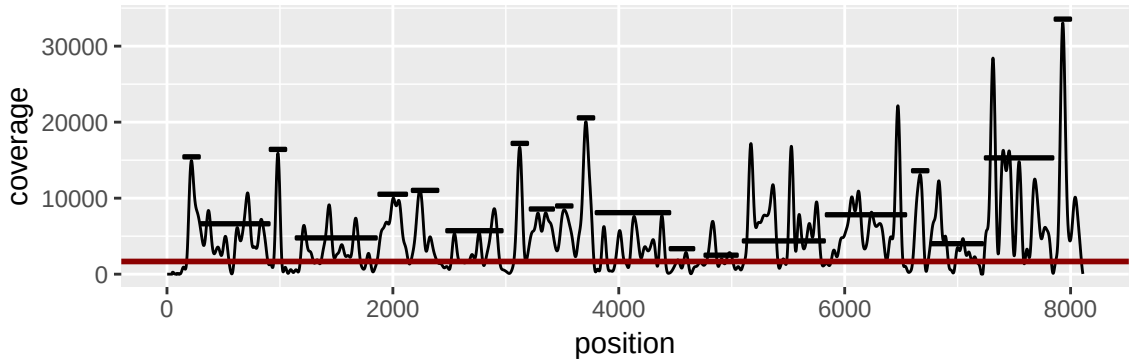

Figure 9: Simple example of ranges manipulation to plot fuzzy nucleosomes

4 Exporting data

`export.wig` and `export.bed` allow exportation of coverage/intensity values and nucleosome calls in a standard format which works on most of the genome browsers available today (like UCSC Genome Browser or Integrated Genome Browser).

`export.wig` creates WIG files wich are suitable for coverage/intensities, meanwhile `export.bed` creates BED files which contain ranges and scores information, suitable for calls.

5 Generating synthetic maps

nucleR includes a synthetic nucleosome map generator, which can be helpful in benchmarking or comparing data against a random map. `syntheticNucMap` function does that, allowing a full customization of the generated maps.

When generating a map, the user can choose the number of the well-positioned and fuzzy nucleosome, as their variance or maximum number of reads. It also provides an option to calculate the ratio between the generated nucleosome map and a mock control of random reads (like a naked DNA randomly fragmented sample) to simulate hybridation data of Tiling Arrays.

The perfect information about the nucleosome dyads is returned by this function, together with the coverage or ratio profiles.

See the man page of this function for detailed information about the different parameters and options.

```
syn <- syntheticNucMap(wp.num=100, wp.del=10, wp.var=30, fuz.num=20,
    fuz.var=50, max.cover=20, nuc.len=147, lin.len=20, rnd.seed=1,
    as.ratio=TRUE, show.plot=TRUE)
```

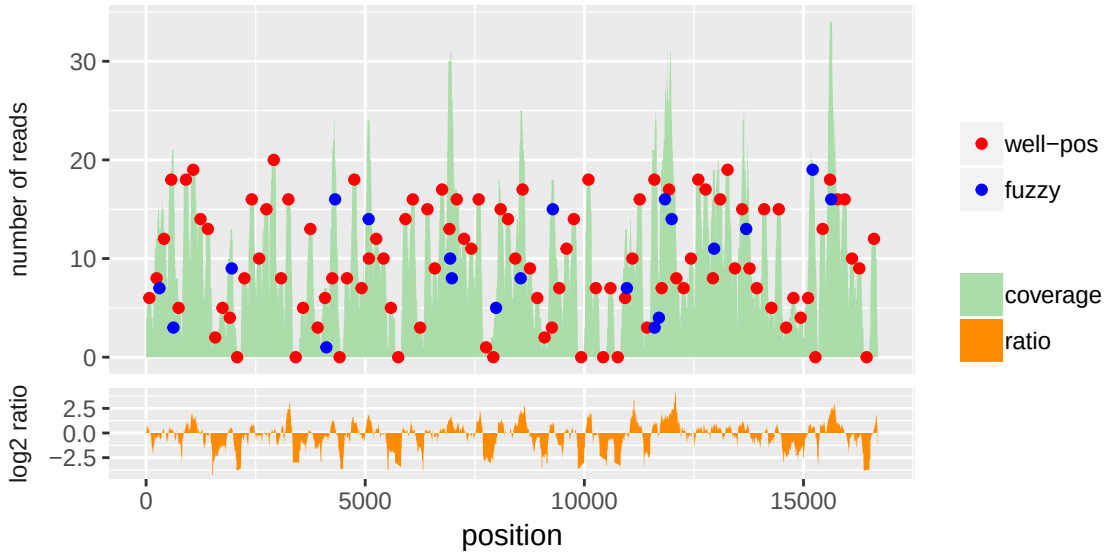

Figure 10: **Example synthetic coverage map of 90 well-positioned (100-10) and 20 fuzzy nucleosomes**

6 References