Quick analysis of nucleosome positioning experiments using the nucleR package

Oscar Flores Guri
Institute for Research in Biomedicine & Barcelona Supercomputing Center
Joint Program on Computational Biology

March 6, 2015

1 Introduction

The nucleR package provides a high-level processing of genomic datasets focused in nucleosome positioning experiments, despite they should be also applicable to chromatin immuno precipitation (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-processing into R/Bioconductor framework.

nucleR works with data from the two main high-throughput 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 importation: 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.[1]

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 [2] is used for Tiling Array experiments as described in Starr [3] and other packages for the Tiling Array manipulation. This kind of experiments can be read with the processTilingArray function.

AlignedRead from package ShortRead [4] 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 [5 6 7], but complex statistical methods were needed for their processing [8].

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 [5 6] or higher order Bayesian Networks [9].

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)).

> require(IRanges)
> library(nucleR)
> 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
[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 Next Generation Sequencing

NGS has become one of the most popular technique to map nucleosome in the genome in the last years [10, 11, 12]. 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 [13], has been recently created to perform preprocessing and quality assessment 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 [6].

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 [14]:

```r
> data(nucleosome_htseq)
> class(nucleosome_htseq)

[1] "RangedData"
attr(.,”package”)
[1] "IRanges"

> nucleosome_htseq

RangedData with 18001 rows and 1 value column across 1 space

   space ranges   strand
     <factor> <IRanges> <character>
     1   chr1  [1, 284]    +
     2   chr1  [5, 205]    +
     3   chr1  [5, 205]    +
     4   chr1  [5, 209]    +
     5   chr1  [5, 283]    +
     6   chr1  [6, 285]    +
     7   chr1  [6, 285]    +
     8   chr1  [8, 126]    +
     9   chr1  [8, 127]    +
    ...
   17993  chr1  [7994, 8148]    +
   17994  chr1  [7994, 8150]    +
   17995  chr1  [7994, 8150]    +
```
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 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).

```r
> # 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

```r
> # 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:

### 2.3 MNase bias correction

The Microccocal Nuclease is a widely used enzyme that has been proved to have a biase for certain dinucleotide steps [14]. In this package we offer a quick way to inspect the effect of such

¹Note 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
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.

```r
> # 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 bias correction. Random nucleosomal and control reads have been generated using `syntheticNucMap` 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 continuous, 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 were used for this task [5, 6, 9, 15, 12]. 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 [16].

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.

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 [17].

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:

```r
> fft_ta = filterFFT(nucleosome_tiling, pcKeepComp=0.01, showPowerSpec=TRUE)
```

![Power Spectrum Figure](image)

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

In the Figure 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 length of the signal, so it has a large value.

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

The approach nucleR follows 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 applied 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, maintaining a high correlation with the original signal but allowing achieve a great performance with a simple peak detection algorithm:

```r
> tiling_raw <- nucleosome_tiling
```
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
> plotPeaks(peaks, htseq_fft, threshold="25%")

![Figure 6: Output of plotPeaks function. Peaks are spotted in red and and detection threshold marked with an horizontal line.](image)

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 appear 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.

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)

<table>
<thead>
<tr>
<th>peak</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9749612</td>
</tr>
<tr>
<td>2</td>
<td>0.6824909</td>
</tr>
<tr>
<td>3</td>
<td>0.2675856</td>
</tr>
<tr>
<td>4</td>
<td>0.3829457</td>
</tr>
<tr>
<td>5</td>
<td>0.4858367</td>
</tr>
<tr>
<td>6</td>
<td>0.8408881</td>
</tr>
</tbody>
</table>

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:

```r
> peaks <- peakDetection(htseq_fft, threshold="25\%", score=TRUE, width=140)
> peaks
```

RangedData with 74 rows and 3 value columns across 1 space

<table>
<thead>
<tr>
<th>space ranges</th>
<th>score</th>
<th>score_w</th>
<th>score_h</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;factor&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;numeric&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>1</td>
<td>[148, 287]</td>
<td>0.8213296</td>
<td>0.6676980</td>
</tr>
<tr>
<td>2</td>
<td>[298, 437]</td>
<td>0.6385168</td>
<td>0.5945427</td>
</tr>
<tr>
<td>3</td>
<td>[382, 521]</td>
<td>0.3157506</td>
<td>0.3639156</td>
</tr>
<tr>
<td>4</td>
<td>[448, 587]</td>
<td>0.5182733</td>
<td>0.6536008</td>
</tr>
<tr>
<td>5</td>
<td>[553, 692]</td>
<td>0.5182437</td>
<td>0.5506507</td>
</tr>
<tr>
<td>6</td>
<td>[645, 784]</td>
<td>0.7229812</td>
<td>0.6050742</td>
</tr>
<tr>
<td>7</td>
<td>[706, 845]</td>
<td>0.2526047</td>
<td>0.2348126</td>
</tr>
<tr>
<td>8</td>
<td>[766, 905]</td>
<td>0.5908013</td>
<td>0.5988045</td>
</tr>
<tr>
<td>9</td>
<td>[913, 1052]</td>
<td>0.9927672</td>
<td>1.0000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>66</td>
<td>[7082, 7221]</td>
<td>0.3244423</td>
<td>0.4674170</td>
</tr>
<tr>
<td>67</td>
<td>[7244, 7383]</td>
<td>0.8948713</td>
<td>0.7897429</td>
</tr>
<tr>
<td>68</td>
<td>[7335, 7474]</td>
<td>0.7430836</td>
<td>0.4978311</td>
</tr>
<tr>
<td>69</td>
<td>[7387, 7526]</td>
<td>0.7495175</td>
<td>0.5112416</td>
</tr>
<tr>
<td>70</td>
<td>[7475, 7614]</td>
<td>0.8645647</td>
<td>0.7558839</td>
</tr>
<tr>
<td>71</td>
<td>[7545, 7684]</td>
<td>0.2120555</td>
<td>0.1762734</td>
</tr>
<tr>
<td>72</td>
<td>[7614, 7753]</td>
<td>0.7874922</td>
<td>0.6540976</td>
</tr>
<tr>
<td>73</td>
<td>[7705, 7844]</td>
<td>0.4716879</td>
<td>0.4552793</td>
</tr>
<tr>
<td>74</td>
<td>[7862, 8001]</td>
<td>0.9135298</td>
<td>0.8270596</td>
</tr>
</tbody>
</table>

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 IRanges functions, such 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:

```r
> nuc_calls <- ranges(peaks[peaks$score > 0.1,])[[1]]
> red_calls <- reduce(nuc_calls)
> red_class <- RangedData(red_calls, isFuzzy=width(red_calls) > 140)
> red_class
```

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

```r
> nuc_calls <- ranges(peaks[peaks$score > 0.1,])[[1]]
> red_calls <- reduce(nuc_calls)
> red_class <- RangedData(red_calls, isFuzzy=width(red_calls) > 140)
> red_class
```

<table>
<thead>
<tr>
<th>space</th>
<th>ranges</th>
<th>isFuzzy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ 148,  287]</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>[ 298,  905]</td>
<td>TRUE</td>
</tr>
<tr>
<td>3</td>
<td>[ 913, 1052]</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>[1143, 1854]</td>
<td>TRUE</td>
</tr>
<tr>
<td>5</td>
<td>[1875, 2122]</td>
<td>TRUE</td>
</tr>
<tr>
<td>6</td>
<td>[2171, 2399]</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>[2476, 2968]</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>[3054, 3193]</td>
<td>FALSE</td>
</tr>
<tr>
<td>9</td>
<td>[3215, 3423]</td>
<td>TRUE</td>
</tr>
<tr>
<td>10</td>
<td>[3472, 3665]</td>
<td>TRUE</td>
</tr>
<tr>
<td>11</td>
<td>[3799, 4452]</td>
<td>TRUE</td>
</tr>
<tr>
<td>12</td>
<td>[4454, 4665]</td>
<td>TRUE</td>
</tr>
<tr>
<td>13</td>
<td>[4762, 5050]</td>
<td>TRUE</td>
</tr>
<tr>
<td>14</td>
<td>[5101, 5821]</td>
<td>TRUE</td>
</tr>
<tr>
<td>15</td>
<td>[5836, 6541]</td>
<td>TRUE</td>
</tr>
<tr>
<td>16</td>
<td>[6599, 6738]</td>
<td>FALSE</td>
</tr>
<tr>
<td>17</td>
<td>[6764, 7221]</td>
<td>TRUE</td>
</tr>
<tr>
<td>18</td>
<td>[7244, 7844]</td>
<td>TRUE</td>
</tr>
<tr>
<td>19</td>
<td>[7862, 8001]</td>
<td>FALSE</td>
</tr>
<tr>
<td>20</td>
<td>[8014, 8200]</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
```
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 which 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.

```r
> 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)
```

References

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


