`htSeqTools`: quality control, visualization and processing high-throughput sequencing data

Evarist Planet [*], Camille Stephan-Otto [*], Oscar Reina [*], Oscar Flores [†], David Rossell [*]

1 Introduction

The `htSeqTools` package provides an easy-to-use toolset to efficiently perform a variety of tasks with high-throughput sequencing data. Although relatively simple-minded, we found the tools to be extremely helpful in quality control assessment, routine pre-processing and analysis steps and in producing useful visualizations. When using the package please cite `?`. The supplementary material of the paper should be a useful resource, as it contains a detailed description of the methods and additional examples (including ChIP-Seq, MNase-Seq and RNA-Seq) with R code.

Emphasis is placed on ChIP-Seq alike experiments, but many functions are also useful for other kinds of sequencing experiments.

Many routines allow performing computations in parallel by specifying an argument `mc.cores`, which uses package `parallel`. As this package is not available in all platforms, in this manual we do not use parallel computing. Please see the help page for each function for more details.

We start by loading the package and a ChIP-Seq dataset which we will use for illustration purposes.

```
> options(width=70)
> library(htSeqTools)
> data(htSample)
> htSample
```

[*]Bioinformatics & Biostatistics Unit, IRB Barcelona
[†]IRB-BSC Joint Research Program on Computational Biology, IRB Barcelona

```
GRangesList object of length 4:
$ctrlBatch1
GRanges object with 102866 ranges and 0 metadata columns:
           seqnames           ranges strand
              <Rle>        <IRanges>  <Rle>
       [1]    chr2L [499167, 499206]      +
       [2]    chr2L [377930, 377969]      -
       [3]    chr2L [306297, 306336]      -
       [4]    chr2L [174413, 174452]      +
       [5]    chr2L [322795, 322834]      +
       ...      ...              ...    ...
  [102862]    chr2L [318650, 318689]      +
  [102863]    chr2L [294283, 294322]      -
  [102864]    chr2L [420010, 420049]      +
  [102865]    chr2L [292888, 292927]      -
  [102866]    chr2L [252943, 252982]      -

...
<3 more elements>
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
> sapply(htSample,nrow)

$ctrlBatch1
NULL

$ipBatch1
NULL

$ctrlBatch2
NULL

$ipBatch2
NULL
```

`htSample` is a `GRangesList` object storing the chromosome, start and end positions for reads mapped to the first 500kb of the drosophila melanogaster chromosome 2L. `htSample` contains 2 immuno-precipitated and 2 control input samples obtained in two separate Illumina sequencing runs, which we named Batch1 and Batch2. The standard Illumina pipeline was used for pre-processing the data. Following the Bowtie defaults, only uniquely mapping reads with at most 2 mismatches in the first 28 bases were kept. We do not give any further details about the experiment, as the results have not yet been published.

You can easily build a `GRangesList` to store multiple `GRanges` objects (coming from different BED files read as data frames for instance). We will extract two elements from htSample in order to simulate a batch of

2 externally loaded samples. Ctrl and IP1 will be two data frames with 'seqnames', 'start', 'end', 'width', and 'strand' columns. Please note that only 'seqnames', 'start' and 'end' are necessary in order to directly generate a `GRanges` object from a data frame. The 'width' and 'strand' column are optional and any additional column will be added to the `GRanges` object as a metadata column.

```
> Ctrl=as.data.frame(htSample[[1]])
> IP1=as.data.frame(htSample[[2]])
> head(Ctrl)

  seqnames  start    end width strand
1    chr2L 499167 499206    40      +
2    chr2L 377930 377969    40      -
3    chr2L 306297 306336    40      -
4    chr2L 174413 174452    40      +
5    chr2L 322795 322834    40      +
6    chr2L 415508 415547    40      +

> makeGRangesFromDataFrame(Ctrl)

GRanges object with 102866 ranges and 0 metadata columns:
           seqnames           ranges strand
              <Rle>        <IRanges>  <Rle>
       [1]    chr2L [499167, 499206]      +
       [2]    chr2L [377930, 377969]      -
       [3]    chr2L [306297, 306336]      -
       [4]    chr2L [174413, 174452]      +
       [5]    chr2L [322795, 322834]      +
       ...      ...              ...    ...
  [102862]    chr2L [318650, 318689]      +
  [102863]    chr2L [294283, 294322]      -
  [102864]    chr2L [420010, 420049]      +
  [102865]    chr2L [292888, 292927]      -
  [102866]    chr2L [252943, 252982]      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> htSample2 <- GRangesList(Ctrl=makeGRangesFromDataFrame(Ctrl),
+                          IP1=makeGRangesFromDataFrame(IP1))
> htSample2

GRangesList object of length 2:
$Ctrl
GRanges object with 102866 ranges and 0 metadata columns:
           seqnames           ranges strand
              <Rle>        <IRanges>  <Rle>
       [1]    chr2L [499167, 499206]      +
       [2]    chr2L [377930, 377969]      -
       [3]    chr2L [306297, 306336]      -
```

```
       [4]    chr2L [174413, 174452]      +
       [5]    chr2L [322795, 322834]      +
       ...      ...                ...    ...
  [102862]    chr2L [318650, 318689]      +
  [102863]    chr2L [294283, 294322]      -
  [102864]    chr2L [420010, 420049]      +
  [102865]    chr2L [292888, 292927]      -
  [102866]    chr2L [252943, 252982]      -

...
<1 more element>
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
> htSample2[['Ctrl']]
```

```
GRanges object with 102866 ranges and 0 metadata columns:
           seqnames            ranges strand
              <Rle>         <IRanges>  <Rle>
       [1]    chr2L [499167, 499206]      +
       [2]    chr2L [377930, 377969]      -
       [3]    chr2L [306297, 306336]      -
       [4]    chr2L [174413, 174452]      +
       [5]    chr2L [322795, 322834]      +
       ...      ...                ...    ...
  [102862]    chr2L [318650, 318689]      +
  [102863]    chr2L [294283, 294322]      -
  [102864]    chr2L [420010, 420049]      +
  [102865]    chr2L [292888, 292927]      -
  [102866]    chr2L [252943, 252982]      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

2 Quality control

2.1 A PCA analogue for sequencing data

PCA is a commonly used technique to assess overall quality and identify problematic samples in high-throughput experiments. PCA requires to define a common set of entities (e.g. genes) for all samples and obtain some numerical measurement for each entity in each sample (e.g. gene expression). Therefore, unfortunately PCA is not directly applicable to sequencing data. One option is to pre-process the data so that PCA can be applied, e.g. computing the number of reads falling in some pre-defined genomic regions. The inconvenient of this approach is its lack of generality, since different kinds of sequencing data generally require different pre-processing. For instance,

while in RNA-Seq we can obtain a PCA based on RPKM expression measures (**?**), this same strategy is not adequate for ChIP-Seq data where many reads may be mapped to promoter or inter-genic regions.

Instead, we propose comparing the read coverage across samples and using Multi-Dimensional Scaling (MDS) to obtain a low-dimensional visual representation. Read coverage is a universal measure which can be computed efficiently for any type of sequencing data. We measure similarity between samples i and j with ρ_{ij}, the correlation between their log-coverages, and define their distance as $d_{ij} = 0.5(1 - \rho_{ij})$. Pearson, Spearman and Kendall correlation coefficients are available. We compute correlations in the log-scale to take into account that the coverage distribution is typically highly asymmetric. In principle, Spearman is more general as it captures non-linear associations, but in practice all options typically produce very similar results. MDS is then used to plot the samples in a low-dimensional space, in a way such that the Euclidean distance between two points is closest to the Pearson distances. Our approach is implemented in a `cmds` method for `GRangesList` objects. We illustrate its use by obtaining a two-dimensional MDS for our sample data.

```
> cmds1 <- cmds(htSample,k=2)

Computing coverage...
Computing correlations...

> cmds1

Object of class cmdsFit approximating distances between 4 objects
R-squared= 1
```

The R^2 coefficient between the original distances and their approximation in the plot can be seen as an analogue to the percentage of explained variability in a PCA analysis. For our sample data R^2=1 (up to rounding), which indicates a perfect fit and that therefore a 3-dimensional plot is not necessary.

```
> plot(cmds1)
```

Figure **??** shows the resulting plot. The IP samples from both runs group nicely, indicating that they have similar coverage profiles. The control samples also group reasonably well, although they present more differences than the IP samples. This is to be expected, since the IP samples focus on a relatively small genomic regions. The MDS plot also reveals a minor batch effect, as samples from the same batch appear slightly closer in the map.

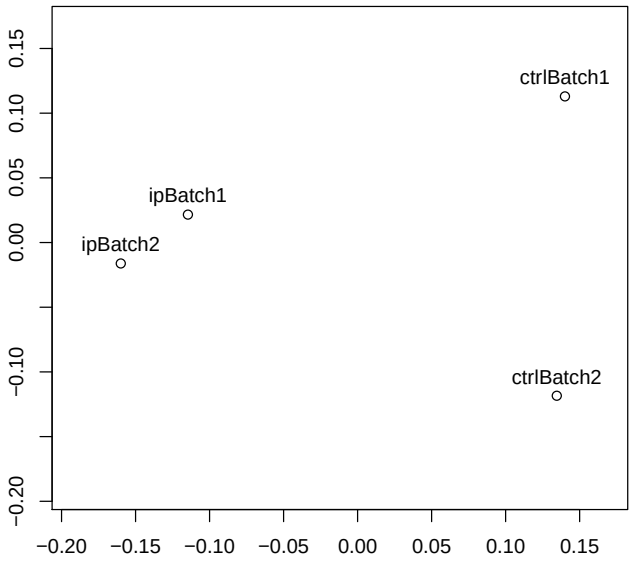

Figure 1: 2-dimensional MDS plot. Samples with similar coverage appear close-by

2.2 Exploring coverage uniformity

In some next-generation sequencing experiments we expect some samples to exhibit large accumulations of reads in certain genomic regions, whereas other samples should present a more uniform coverage along the genome. For instance, in ChIP-seq one should observe higher peaks for immuno-precipitated (IP) samples than in the controls. That is, IP samples should present coverage rich and coverage poor regions, whereas differences in coverage in the controls should be smaller.

In these cases we propose to measure the unevennes in the coverage using either the coverage standard deviation or Gini's coefficient (**?**), which is a classical econometrics measure to assess unevennes of wealth distribution. Comparing these statistical dispersion measures between samples can reveal samples with innefficient immuno-precipitation (e.g. due to an inadequate antibody). Both measures can be easily obtained with the functions `ssdCoverage` and `giniCoverage`. Simple algebra shows that the expected value of the coverage standard deviation is proportional to $\sqrt{n}$, where n is the number of reads. Therefore `ssdCoverage` reports $\mathrm{SD}_n = \mathrm{SD}/\sqrt{n}$ as a measure that can be compared across samples with different number of reads. Similarly, simulations show that the expected Gini also depends on n. Since no closed-form expression is available, we estimate its expected value by generating n reads uniformly distributed along the genome and computing the Gini coefficient. The adjusted Gini (G_n) is the difference between the observed Gini (G) minus its estimated expected value $\hat{\mathrm{E}}(G|n)$.

```
> ssdCoverage(htSample)

ctrlBatch1   ipBatch1 ctrlBatch2   ipBatch2
  55.98648  169.15785   18.14517  100.43349

> giniCoverage(htSample,mc.cores=1)

Simulating uniformily distributed data
Calculating gini index of original data
Simulating uniformily distributed data
Calculating gini index of original data
Simulating uniformily distributed data
Calculating gini index of original data
Simulating uniformily distributed data
Calculating gini index of original data
               gini gini.adjust
ctrlBatch1 0.8038884   0.6098682
ipBatch1   0.9378820   0.7958104
ctrlBatch2 0.4813761   0.2274161
ipBatch2   0.9127399   0.7132966
```

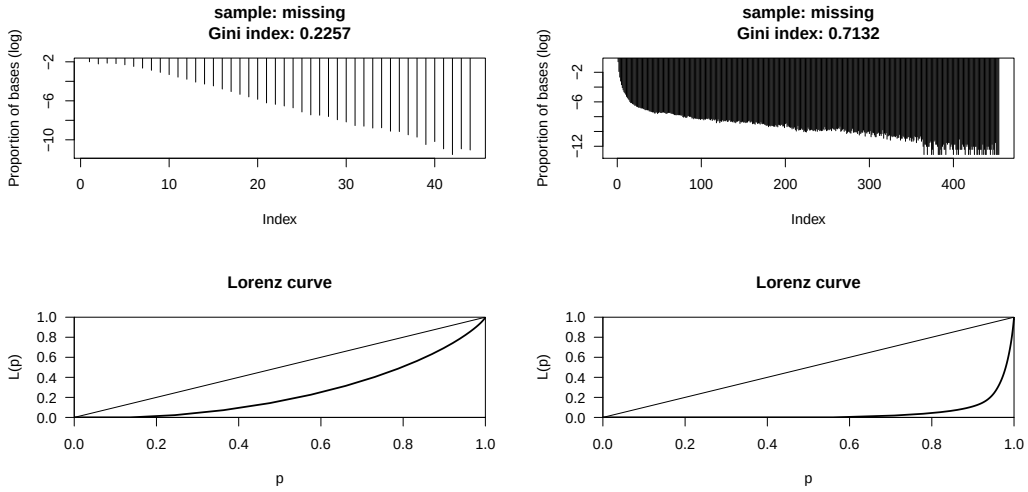

Figure 2: Lorenz curves to assess coverage uniformity. Left: control; Right: immuno-precipitated sample

The coverage exhibits higher dispersion in the IP samples than in the controls, which indicates there were no substantial problems in the immuno-precipitation. The function `giniCoverage` allows to graphically assess the non-uniformity in the coverage distribution by plotting the probability and cumulative probability function. The top panels in Figure **??** show the log probability mass function of the coverage, and the bottom panels show the Lorenz curve (**?**, see `Lc` from package `ineq` for details). We observe a more pronounced non-uniformity in the IP sample.

```
> giniCoverage(htSample[['ctrlBatch2']],mc.cores=1,mk.plot=TRUE)
```

```
> giniCoverage(htSample[['ipBatch2']],mc.cores=1,mk.plot=TRUE)
```

2.3 Detecting over-amplification artifacts

High-throughput sequencing requires a PCR amplification step to enrich for adapter-ligated fragments. This step can induce biases as some DNA regions amplify more efficiently than others (e.g. depending on GC content). These PCR artifacts, caused by over-amplification or primer dimers, affect the accuracy of the coverage and can create biases in the downstream analyses. The function `filterDupReads` aims to automatically detect and remove these artifacts. The basic rationale is that, by counting the number of times that each read is repeated, we can detect the reads repeating an unusually

large number of times. The argument `maxRepeats` can be used to eliminate all reads appearing more than this user-specified threshold. However, notice that ideally this threshold should be determined for each sample separately, as the expected number of naturally occurring repeats depends on the sequencing depth, and may also depend on the characteristics of each sample. For instance, sequences from IP samples focus on a relatively small genomic region while those from controls are distributed along most of the genome, and therefore we expect a higher number of repeats in IP samples. When specifying the argument `fdrOverAmp`, `filterDuplReads` determines the threshold in a data-adaptive manner.

Although this filtering can be performed with a single call to `filterDuplReads`, we now illustrate its inner workings in a step-by-step fashion. We add 200 repeats of an artificial sequence to sample `"ctrlBatch1"`, and count the number of times that each sequence appears with the function `tabDuplReads`.

```
> contamSample <- GRanges('chr2L',IRanges(rep(1,200),rep(36,200)),strand='+')
> contamSample <- c(htSample[['ctrlBatch1']],contamSample)
> nrepeats <- tabDuplReads(contamSample)
> nrepeats

ans
    1     2     3     4     5     6     7     8     9    10    11
11812 10112  6744  4083  2325  1343   727   447   212   113    87
   12    13    14    15    16    17    18    19    20    22   200
   57    35    19     9     9     4     5     1     2     1     1
```

There are 11812 sequences appearing only once, 10112 appearing twice etc. The function `fdrEnrichedCounts` (called by `filterDuplReads`) determines the over-amplification threshold in a data-adaptive manner. Basically, it assumes that only large number of repeats are artifacts and models the reads with few repeats with a truncated negative binomial mixture (fit via Maximum Likelihood), which we observed to fit experimental data reasonably well. The number of components to be used is chosen in parameter `components`. If this parameter has value 0 the optimal number of components is selected using the Bayesian information criterion (BIC). Here we used one component for computational speed. An empirical Bayes approach similar to that in **?** is then used to estimate the FDR associated with a given cutoff (see `help(fdrEnrichedCounts)` for details).

```
> q <- which(cumsum(nrepeats/sum(nrepeats))>.999)[1]
> q

14
14
```

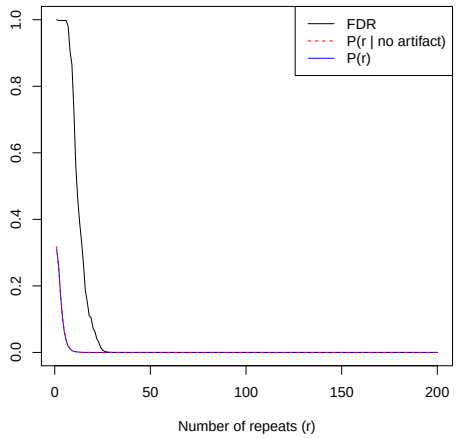

Figure 3: Black line: estimated FDR for each number of repeats cutoff. Red line: distribution of the number of repeats as estimated by a truncated negative binomial with 2 components (representing not over-amplified reads). Blue line: distribution of the number of repeats in the observed data.

```
> fdrest <- fdrEnrichedCounts(nrepeats, use=1:q, components=1)
> numRepeArtif <- rownames(fdrest[fdrest$fdrEnriched<0.01,])[1]
> numRepeArtif

[1] "25"
```

Here we assumed that less than $1/1000$ reads are artifacts and fit a negative binomial truncated at $[1, 14]$. Notice that, although here we used `fdrEnrichedCounts` to detect over-amplification, it can also be used in any other setup when one whishes to detect large counts. In Figure **??** we produce a plot showing the estimated FDR for a series of cutoffs. The argument `fdrOverAmp` to `filterDuplReads` indicates the FDR cutoff to determine over-amplification. We also show the distribution of the observed number of repeats (blue) and its truncated negative binomial approximation in $[1, 14]$ (red). Notice that any sequence repeating over 25 times (including our artificial sequence repeating 200 times) is regarded as an artifact.

```
> plot(fdrest$fdrEnriched,type='l',ylab='',xlab='Number of repeats (r)')
> lines(fdrest$pdfOverall,col=4)
> lines(fdrest$pdfH0,col=2,lty=2)
> legend('topright',c('FDR','P(r | no artifact)','P(r)'),lty=c(1,2,1),col=c(1,2,4))
```

3 Data pre-processing

We discuss several tools which can be useful for ChIP-seq data preprocessing. In these studies there typically is a strand-specific bias: reads coming from the + strand pile up to the left of reads from the - strand. Removing this bias is important, as it provides a highly increased accuraccy for peak detection. `alignPeaks` implements a procedure to correct this bias, which is fairly similar to the MACS procedure (**?**). A nice alternative is provided in function `estimate.mean.fraglen` from package `chipseq`. To illustrate the need for the adjustment we plot the coverage in a certain genomic region before the adjustment (displayed in Figure **??**, left panel).

```
> covbefore <- coverage(htSample[['ipBatch2']])
> covbefore <- window(covbefore[['chr2L']],295108,297413)
> plot(as.integer(covbefore),type='l',ylab='Coverage')
```

Now we perform the adjustment with `alignPeaks` and plot the resulting coverage as the solid black line in Figure **??** (right panel). In blue and red color we display the coverage computed separately from reads on the + and - strands, respectively. The blue and red lines present a similar profile, but they are shifted. Exploring other peaks reveals similar patterns. Removing this strand specific bias results in sharper peaks and prevents detecting two separate peaks when there should actually be one, as illustrated by the leftmost peak in Figure **??**.

```
> ip2Aligned <- alignPeaks(htSample[['ipBatch2']],strand='strand', npeaks=100)

Estimated shift size is 61.49423

> covafter <- coverage(ip2Aligned)
> covafter <- window(covafter[['chr2L']],295108,297413)
> covplus <- coverage(htSample[['ipBatch2']][strand(htSample[['ipBatch2']])=='+'])
> covplus <- window(covplus[['chr2L']],295108,297413)
> covminus <- coverage(htSample[['ipBatch2']][strand(htSample[['ipBatch2']])=='-'])
> covminus <- window(covminus[['chr2L']],295108,297413)

> plot(as.integer(covafter),type='l',ylab='Coverage')
> lines(as.integer(covplus),col='blue',lty=2)
> lines(as.integer(covminus),col='red',lty=2)
```

In ChIP-seq experiments, it is sometimes convenient to extend the reads to take into account that we only sequenced the first few bases (typically 40-100 bp) from a larger DNA fragment (typically around 300bp). In practice, this achieves some smoothing of the read coverage. `extendRanges` extends the reads up to a user-specified length.

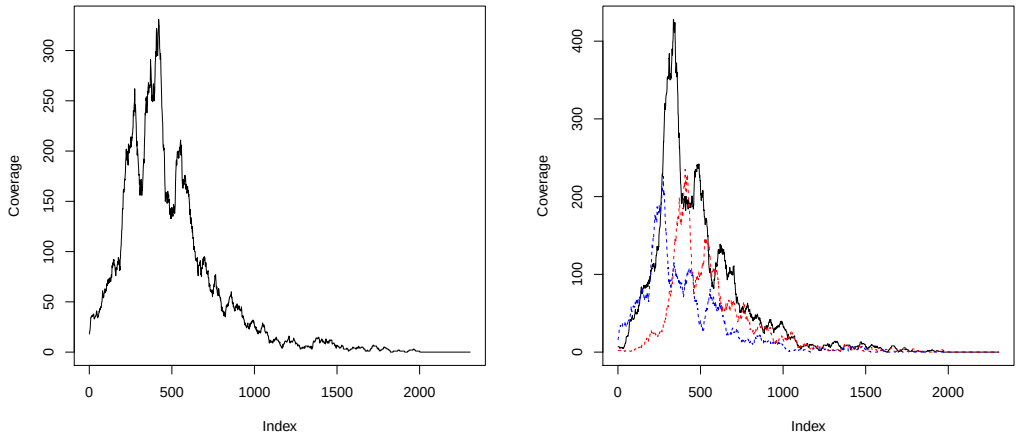

Figure 4: Coverage for gene p38b. Left: Before adjustment; Right: After adjustment. Blue line: + strand; Red line: - strand; Black line: global estimate after peak alignment by `alignPeaks`.

4 Basic data analysis

Finding genomic regions with large accumulation of reads is an important task in many sequencing experiments, including ChIP-Seq and RNA-Seq. `islandCounts` performs a simple analysis using tools provided in the `IRanges` package. We search for genomic regions with an overall coverage ≥ 10 (i.e. across all samples), and obtain the number of reads in each sample overlapping with these regions.

```
> ip <- c(htSample[[2]],htSample[[4]])
> ctrl <- c(htSample[[1]],htSample[[3]])
> pool <- GRangesList(ip=ip, ctrl=ctrl)
> counts <- islandCounts(pool,minReads=10)
> head(counts)
```

```
GRanges object with 6 ranges and 2 metadata columns:
      seqnames       ranges strand |        ip      ctrl
         <Rle>    <IRanges>  <Rle> | <integer> <integer>
  [1]    chr2L [5191, 5211]      * |         7         5
  [2]    chr2L [5214, 5227]      * |         9         4
  [3]    chr2L [5411, 5476]      * |        15        14
  [4]    chr2L [5484, 5484]      * |         5         5
  [5]    chr2L [5602, 5645]      * |        13        13
  [6]    chr2L [5650, 5652]      * |         7         3
```

```
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

There are a number of analysis strategies to compare these counts between groups. For instance for short RNA sequencing data we could compare expression levels across groups using the tools in package `DEseq`. Here we show a simple analysis based on likelihood-ratio tests with the function `enrichedRegions`.

```
> mappedreads <- c(ip=nrow(ip),ctrl=nrow(ctrl))
> mappedreads

NULL

> regions <- enrichedRegions(sample1=ip,sample2=ctrl,minReads=10,mappedreads=mappedreads,p.adj
> head(regions)

GRanges object with 6 ranges and 5 metadata columns:
      seqnames         ranges strand |   sample1   sample2
         <Rle>      <IRanges>  <Rle> | <integer> <integer>
  [1]    chr2L [66674, 67923]      * |       651       249
  [2]    chr2L [72428, 73199]      * |      2044       200
  [3]    chr2L [73357, 74080]      * |       991       144
  [4]    chr2L [74484, 74484]      * |        10         0
  [5]    chr2L [86284, 86285]      * |        10         0
  [6]    chr2L [86294, 87202]      * |       898       260
                   pvalue            rpkm1            rpkm2
                <numeric>        <numeric>        <numeric>
  [1] 8.07214328893402e-05 1767.67676767677 1224.52743199631
  [2]                    0 8986.60120622347 1592.54561249702
  [3]                    0  4645.8690751152 1222.65269785959
  [4]   0.0431594471248686 33941.5661996307                0
  [5]   0.0431594471248686 16970.7830998154                0
  [6]                    0 3353.08321752127 1758.28248262048
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

> nrow(regions)

NULL
```

The argument `twoTailed=FALSE` indicates that only regions with a significantly higher number of reads in `sample1` than in `sample2` are reported. Regions with overall coverage $\geq$10 are selected, and a likelihood-ratio test is used to compare the proportion of reads in `sample1` falling in a given region with the proportion in `sample2`. When setting `exact` to `TRUE`, a permutation-based Chi-square test is used whenever the expected counts in any group is below 5. Here we reported only regions with Benjamini-Yekutielli adjusted p-values below 0.05.

`enrichedRegions` can also be used with no control sample, in which case it looks for islands with a significant accumulation of reads with an exact Binomial test. The null hypothesis assumes that a read is equally likely to come from any of the selected regions.

A related function is `enrichedPeaks`, which can be used to find peaks within the enriched regions. Peaks are defined as enriched regions where the difference in coverage between `sample1` and `sample2` is above a user-specified threshold. In this example we use `minHeight=100`.

```
> peaks <- enrichedPeaks(regions, sample1=ip, sample2=ctrl, minHeight=100)
> peaks

GRanges object with 162 ranges and 2 metadata columns:
        seqnames            ranges strand |    height region.pvalue
           <Rle>         <IRanges>  <Rle> | <integer>     <numeric>
    [1]    chr2L   [72615, 72665]      * |       141             0
    [2]    chr2L   [72668, 72940]      * |       233             0
    [3]    chr2L   [72988, 72994]      * |       114             0
    [4]    chr2L   [72996, 72997]      * |       100             0
    [5]    chr2L   [73475, 73475]      * |       100             0
    ...      ...              ...    ... .       ...           ...
  [158]    chr2L [491339, 491341]      * |       102             0
  [159]    chr2L [491345, 491347]      * |       104             0
  [160]    chr2L [491349, 491349]      * |       102             0
  [161]    chr2L [491354, 491379]      * |       113             0
  [162]    chr2L [491382, 491384]      * |       103             0
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

It is possible to merge nearby regions, e.g. say no more than 300bp apart, into a single region with the function `mergeRegions`. `mergeRegions` allows to combine a numerical score across regions with any user-defined function, e.g. the mean or median. In the following example we merge regions less than 300bp apart and compute their median `'height'`.

```
> mergeRegions(peaks, maxDist=300, score='height', aggregateFUN='median')

GRanges object with 35 ranges and 1 metadata column:
       seqnames            ranges strand |    height
          <Rle>         <IRanges>  <Rle> | <numeric>
   [1]    chr2L [ 72615,  72997]      * |     127.5
   [2]    chr2L [ 73475,  73712]      * |       102
   [3]    chr2L [102368, 103198]      * |     935.5
   [4]    chr2L [106082, 106784]      * |      1272
   [5]    chr2L [108119, 109322]      * |       108
   ...      ...              ...    ... .       ...
  [31]    chr2L [431278, 432063]      * |     116.5
  [32]    chr2L [453202, 453882]      * |       531
```

```
  [33]    chr2L [473207, 473721]      * |       508
  [34]    chr2L [478840, 479586]      * |     342.5
  [35]    chr2L [491110, 491384]      * |     103.5
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

A common task is to identify genomic features close to the identified regions/peaks, e.g. finding the closest gene. This can be performed with the function `annotatePeakInBatch` from package `ChIPpeakAnno`, for instance. Sometimes it is of interest to compare the list of genes which had a nearby peak with the genes found in another experiment. The function `listOverlap` quantifies the overlap and tests for its statistical significance.

Another analysis which may be of interest is locating hot chromosomal regions, i.e., regions in the chromosome with a large number of peaks. The function `enrichedChrRegions` can be used for this purpose. First we need to define a named vector indicating the chromosome lengths. Since our example data only contains reads aligning to the first 500,000 bases of chr2L, we manually its length. More generally, one can determine the chromosome lengths from Bioconductor packages (e.g. for drosophila melanogaster load `BSgenome.Dmelanogaster.UCSC.dm3` and evaluate `seqlengths(Dmelanogaster)`, and similarly for other organisms). We run the function setting a window size of 9999 base pairs and a 0.05 false discovery rate level. For computational speed here we only use `nSims=1` simulations to estimate the FDR.

```
> chrLength <- 500000
> names(chrLength) <- c('chr2L')
> chrregions <- enrichedChrRegions(peaks, chrLength=chrLength, windowSize=10^4-1, fdr=0.05, nS
> chrregions

GRanges object with 11 ranges and 0 metadata columns:
       seqnames           ranges strand
          <Rle>        <IRanges>  <Rle>
   [1]    chr2L [ 68708,  77639]      *
   [2]    chr2L [104266, 113558]      *
   [3]    chr2L [150732, 161683]      *
   [4]    chr2L [203117, 212706]      *
   [5]    chr2L [244550, 255472]      *
   [6]    chr2L [272320, 276407]      *
   [7]    chr2L [277654, 282740]      *
   [8]    chr2L [292857, 301000]      *
   [9]    chr2L [301160, 302862]      *
  [10]    chr2L [413952, 419140]      *
  [11]    chr2L [486367, 500000]      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

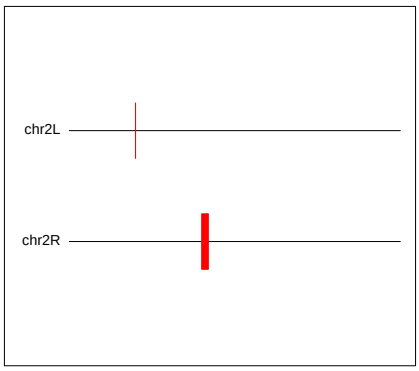

Figure 5: Chromosomal regions with a large number of hits. Red marks indicate regions with high concentration of peaks.

`enrichedChrRegions` returns a `GRanges` which for our sample data is empty, suggesting that there is no region with an unusually high density of enriched regions. Two related functions are `countHitsWindow` which computes the moving average of the number of hits, and `plotChrRegions` to visualize the results. For illustrative purposes we make up two regions with high density of enriched peaks and plot them.

```
> chrregions <- GRanges(c('chr2L','chr2R'), IRanges(start=c(100000,200000),end=c(100100,210000
> plotChrRegions(regions=chrregions, chrLength=c(chr2L=500000,chr2R=500000))
```

5 Plots

`stdPeakLocation` and `PeakLocation` produce a plot useful for exploring overall patterns in ChIP-chip or ChIP-seq experiments. Basically, it creates a density plot indicating where the peaks were located with respect to the closest gene/genomic feature. `stdPeakLocation` indicates the location in standardized coordinates, i.e. relative to each gene/features's length, which in some situations can help making genes/features comparable. `PeakLocation` produces the same plot in base pairs (i.e. non-standardized coordinates), which is useful as distances have a direct physical interpretation, e.g. to relate the peak location with nucleosome positioning. As mentioned earlier, function `annotatePeakInBatch` from package `ChIPpeakAnno` can be

```
NULL
```

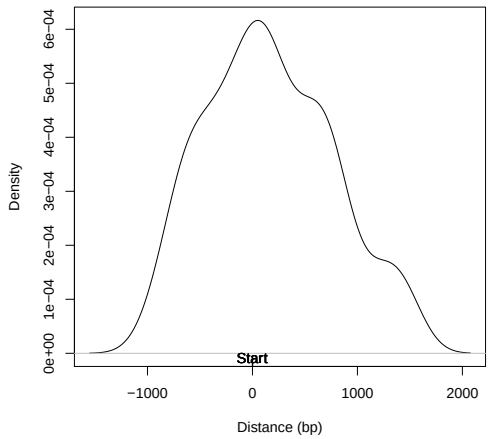

Figure 6: Distribution of peaks around the closest gene

used to find the gene closest to each region. For illustrative purposes here we assign a fake gene to each region. The fake genes start, end, strand and distance from the start to the region by default are assumed to be stored in `'start_position'`, `'end_position'`, `'strand'` and `'distance'`, respectively (although different names can be given as arguments to `PeakLocation` and `stdPeakLocation`).

```
> set.seed(1)
> peaksanno <- peaks
> mcols(peaksanno)$start_position <- start(peaksanno) + runif(length(peaksanno),-500,1000)
> mcols(peaksanno)$end_position <- mcols(peaksanno)$start_position + 500
> mcols(peaksanno)$distance <- mcols(peaksanno)$start_position - start(peaksanno)
> strand(peaksanno) <- sample(c('+','-'),length(peaksanno),replace=TRUE)
> PeakLocation(peaksanno,peakDistance=1000)
```

```
NULL
```

Figure **??** shows the resulting plot. We see that most of the peaks occur right around the transcription start site.

Two related functions are `regionsCoverage` and `gridCoverage`, which evaluate the coverage on user-specified genomic regions. We illustrate their use by obtaining the coverage for the regions which we found to be enriched (as previously described). `regionsCoverage` computes the coverage in the specified regions. As each region has a different length it may be hard to

compare coverages across regions, e.g. to cluster regions with similar coverage profiles. `gridCoverage` simplifies this task by evaluating the coverage on a regular grid of 500 equally spaced points between the region start and end. The result is stored in an object of class `gridCover`. The object contains the coverage, which can be accesed with the method `getViewsInfo`.

```
> cover <- coverage(ip)
> rcov <- regionsCoverage(seqnames(regions),start(regions),end(regions),cover=cover)
> names(rcov)

[1] "views"     "viewsInfo"

> rcov[['views']]

RleViewsList of length 1
names(1): chr2L

> gridcov <- gridCoverage(rcov)
> dim(getCover(gridcov))

[1]  50 500

> getViewsInfo(gridcov)

DataFrame with 50 rows and 3 columns
      strand   meanCov    maxCov
    <factor> <numeric> <integer>
1          +  20.64080        51
2          + 105.58161       244
3          +  54.57182       129
4          +  10.00000        10
5          +  10.00000        10
...      ...       ...       ...
46         + 216.27010      1033
47         + 110.71647       575
48         +  13.15815        41
49         + 112.99047       585
50         +  53.23333       187
```

We plot the coverage for the selected regions and see that they present different profiles Figure **??**, which suggests the use of some clustering technique to find subgroups of regions behaving similarly.

```
> ylim <- c(0,max(getViewsInfo(gridcov)[['maxCov']]))
> plot(gridcov, ylim=ylim,lwd=2)
> for (i in seq_along(regions)) lines(getCover(gridcov)[i,], col='gray', lty=2)
```

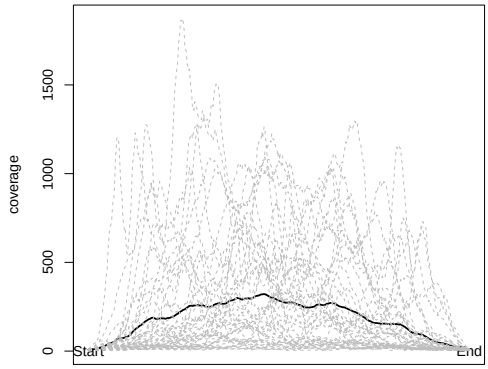

Figure 7: Coverage for some selected regions