ChIP-seq analysis basics

Aleksandra Pękowska, Simon Anders

June 15, 2015

Contents

1 Introduction

This vignette describes several basic steps in the analysis of ChIP-seq for histone modification - here H3K27 acetylation (H3K27ac).

1.1 Aims of the tutorial

The aim of the present lab is to show the reader how to:
1. Read ChIP-seq experiment to *R*
2. Extend the reads and bin the data
3. Create .bedGraph files for data sharing
4. Visualize ChIP-seq files with *R*
5. Perform basic analysis of ChIP-seq peaks
6. Generate average profiles and heatmaps of ChIP-seq enrichment around a set of genomic loci

In the appendix part, we show how to download, preprocess and asses the quality of .fastq files.

2 Data

H3K27ac is a histone modification associated with active promoters and enhancers. We downloaded data corresponding to a ChIP-seq experiment mapping the H3K27ac histone modification in two replicates of mouse Embryonic Stem cells (mES) along with the input control sample from the study *Histone H3K27ac separates active from poised enhancers and predicts developmental state* by Creyghton *et al.*, PNAS 2010.

To get started quickly, we here describe the initial preprocessing only briefly. See the Appendix for full details.

2.1 Preprocessing of data

The first part of ChIP-seq analysis workflow consists in read preprocessing. We will not focus here on these first steps, we outline them and provide the code in the *Appendix* part of the vignette. The three major steps in the preprocessing are briefly outlined below.

Initial quality assessment

Sequenced reads are saved in .fastq files. The very first step in the analyses of sequencing results consists in quality assessment. The *R* package *ShortRead* provides a *qa* to perform this analysis. The reader can find the code necessary to generate a *HTML* read quality control report in the *Appendix* part of the vignette.

External to *R* data opperations

Initial parts of the analysis of sequenced reads include: alignment, filtering and peak finding. They can be performed using tools such as *Bowtie2*, *samtools* or *MACS*. We provide all the necessary code in the *Appendix* part of the vignette.

Additional considerations

Visualisation and read distribution analysis parts of this vignette. They require *biomart* database querying via the internet. We hence provide the necessary objects in the package. Code for their generation is found in the *Appendix* part of the vignette.

To reduce memory requirements, we focus on filtered reads mapping only to chromosome 6.

2.2 Data package

To save time, we have performed we above steps already for you and placed the produced files and R objects in a data package called *EpigeneticsCSAMA2015*, which we load now. (Note that such a data package is used for convenience in this course, but typically, you would not package up interemediate data in this way.)

```
library(EpigeneticsCSAMA2015)
dataDirectory =  system.file("bedfiles", package="EpigeneticsCSAMA2015")
```

The variable *dataDirectory* shows where the files that we will read in the following are on your computer.

```
dataDirectory
```

```
## [1] "/home/anders/R/x86_64-pc-linux-gnu-library/3.2/EpigeneticsCSAMA2015/bedfiles"
```

Have a look at them with a text editor.

3 Reading the filtered ChIP-seq reads

We need to load the *GenomicRanges*, *rtracklayer* and *IRanges* packages. To read the .bam file to *R*, we use the *import.bed* function from the *rtracklayer* package. The result is a *GRanges* object. This is an extremely useful and powerful class of objects which the readers are already familiar with. Each filtered read is represented here as a genomic interval.

```
library(GenomicRanges)
library(rtracklayer)
library(IRanges)

input = import.bed(file.path(dataDirectory, 'ES_input_filtered_ucsc_chr6.bed'),
                   asRangedData=FALSE)
rep1 = import.bed(file.path(dataDirectory, 'H3K27ac_rep1_filtered_ucsc_chr6.bed'),
                  asRangedData=FALSE)
rep2 = import.bed(file.path(dataDirectory, 'H3K27ac_rep2_filtered_ucsc_chr6.bed'),
                  asRangedData=FALSE)
```

The objects *input*, *rep1* and *rep2* hold the genomic annotation of the filtered reads for the input sample and ChIP-seq replicate 1 and replicate 2, respectively. We display the *rep1* object. We see that the strand information, read name along with alignment score are included as information for each read.

```
rep1
```

```
## GRanges object with 481412 ranges and 2 metadata columns:
##            seqnames                ranges strand   |                name
##               <Rle>             <IRanges>  <Rle>   |         <character>
##        [1]     chr6 [ 85222462,  85222497]      -   |       SRR066766.303
##        [2]     chr6 [134189439, 134189474]      +   |       SRR066766.374
##        [3]     chr6 [ 47920826,  47920861]      +   |       SRR066766.393
##        [4]     chr6 [143565148, 143565183]      +   |       SRR066766.397
##        [5]     chr6 [ 39392692,  39392727]      -   |       SRR066766.438
##        ...      ...                   ...    ... ...                 ...
##   [481408]     chr6 [ 86800209,  86800244]      -   | SRR066766.14657325
##   [481409]     chr6 [ 91422497,  91422532]      -   | SRR066766.14657340
##   [481410]     chr6 [ 54433776,  54433811]      +   | SRR066766.14657362
```

```
##   [481411]     chr6 [120020286, 120020321]      -   | SRR066766.14657382
##   [481412]     chr6 [ 85113322,  85113357]      -   | SRR066766.14657387
##               score
##           <numeric>
##       [1]        40
##       [2]        42
##       [3]        42
##       [4]        42
##       [5]        42
##       ...       ...
##   [481408]        42
##   [481409]        42
##   [481410]        42
##   [481411]        42
##   [481412]        42
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We see that we have roughly the same number of reads in the input and IP-ed experiments.

```
length(input)

## [1] 465823

length(rep1)

## [1] 481412

length(rep2)

## [1] 506287
```

4 Preparation of the ChIP-seq and control samples: read extension

The reads correspond to sequences at the end of each IP-ed fragment (single-end sequencing data). As discussed in the lecture, we need to extend them to represent each IP-ed DNA fragment.

We estimate the mean read length using the *estimate.mean.fraglen* function from *chipseq* packege. Next, we extend the reads to the inferred read length using the *resize* function. We remove any reads for which the coordinates, after the extension, exceed chromosome length. These three analysis steps are wrapped in a single function *prepareChIPseq* function which we define below.

```
library(chipseq)

prepareChIPseq = function(reads){

    frag.len = median( estimate.mean.fraglen(reads) )
    cat( paste0( 'Median fragment size for this library is ', round(frag.len)))
    reads.extended = resize(reads, width = frag.len)
    return( trim(reads.extended) )
}
```

We next apply it to the input and ChIP-seq samples.

```
input = prepareChIPseq( input )

## Median fragment size for this library is 236

rep1 = prepareChIPseq( rep1 )

## Median fragment size for this library is 122

rep2 = prepareChIPseq( rep2 )

## Median fragment size for this library is 107
```

Compare with above to see how *rep1* has changed.

```
rep1

## GRanges object with 481412 ranges and 2 metadata columns:
##            seqnames                ranges strand   |                name
##               <Rle>             <IRanges>  <Rle>   |         <character>
##        [1]     chr6 [ 85222376,  85222497]      -   |       SRR066766.303
##        [2]     chr6 [134189439, 134189560]      +   |       SRR066766.374
##        [3]     chr6 [ 47920826,  47920947]      +   |       SRR066766.393
##        [4]     chr6 [143565148, 143565269]      +   |       SRR066766.397
##        [5]     chr6 [ 39392606,  39392727]      -   |       SRR066766.438
##        ...      ...                   ...    ... ...                 ...
##   [481408]     chr6 [ 86800123,  86800244]      -   | SRR066766.14657325
##   [481409]     chr6 [ 91422411,  91422532]      -   | SRR066766.14657340
##   [481410]     chr6 [ 54433776,  54433897]      +   | SRR066766.14657362
##   [481411]     chr6 [120020200, 120020321]      -   | SRR066766.14657382
##   [481412]     chr6 [ 85113236,  85113357]      -   | SRR066766.14657387
##               score
##           <numeric>
##        [1]        40
##        [2]        42
##        [3]        42
##        [4]        42
##        [5]        42
##        ...       ...
##   [481408]        42
##   [481409]        42
##   [481410]        42
##   [481411]        42
##   [481412]        42
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

5 Binning the ChIP-seq and control

The next step in the analysis is to count how many reads map to each of the pre-established genomic intervals (bins).

5.1 Generation of bins

We first generate the bins. We will tile the genome into non-overlapping bins of size 200 bp.

To this end we need the information about chromosome sizes in the mouse genome (assembly *mm9*). In the data package, we provide the object *si*, which holds this information. The reader can find the code necessary to create the *si* object in the *Obtaining si object for mm9* of the *Appendix*.

```
data(si)
si
```

```
## Seqinfo object with 21 sequences from mm9 genome:
##   seqnames seqlengths isCircular genome
##   chr1      197195432      FALSE    mm9
##   chr2      181748087      FALSE    mm9
##   chr3      159599783      FALSE    mm9
##   chr4      155630120      FALSE    mm9
##   chr5      152537259      FALSE    mm9
##   ...             ...        ...    ...
##   chr17      95272651      FALSE    mm9
##   chr18      90772031      FALSE    mm9
##   chr19      61342430      FALSE    mm9
##   chrX      166650296      FALSE    mm9
##   chrY       15902555      FALSE    mm9
```

Next, we use the *tileGenome* function from the *GenomicRanges* package to generate a *GRanges* object with intervals covering the genome in tiles (bins) of size of 200 bp.

```
binsize = 200
bins = tileGenome(si['chr6'], tilewidth=binsize,
                  cut.last.tile.in.chrom=TRUE)
bins
```

```
## GRanges object with 747586 ranges and 0 metadata columns:
##            seqnames                ranges strand
##               <Rle>             <IRanges>  <Rle>
##        [1]     chr6           [  1,  200]      *
##        [2]     chr6           [201,  400]      *
##        [3]     chr6           [401,  600]      *
##        [4]     chr6           [601,  800]      *
##        [5]     chr6           [801, 1000]      *
##        ...      ...                   ...    ...
##   [747582]     chr6 [149516201, 149516400]      *
##   [747583]     chr6 [149516401, 149516600]      *
##   [747584]     chr6 [149516601, 149516800]      *
##   [747585]     chr6 [149516801, 149517000]      *
##   [747586]     chr6 [149517001, 149517037]      *
##   -------
##   seqinfo: 1 sequence from mm9 genome
```

5.2 Binning

We now count how many reads fall into each bin. For this purpose, we define the function *BinChIPseq*. It takes two arguments, *reads* and *bins* which are *GRanges* objects.

```
BinChIPseq = function( reads, bins ){

       mcols(bins)$score = countOverlaps( bins, reads )
```

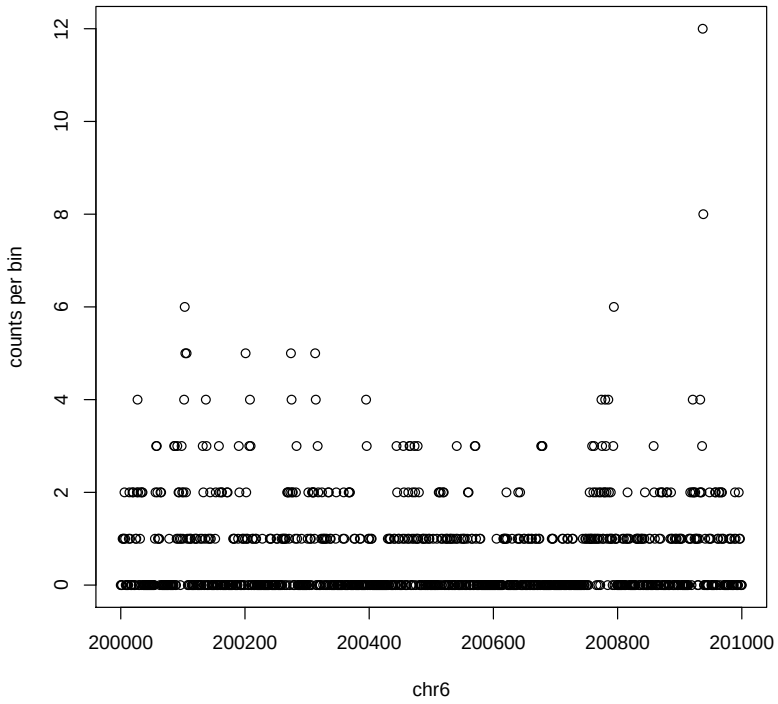
```
       return( bins )
}
```

Now we apply it to the objects *input*, *rep1* and *rep2*. We obtain *input.200bins*, *rep1.200bins* and *rep2.200bins*, which are *GRanges* objects that contain the binned read coverage of the input and ChIP-seq experiments.

```
input.200bins = BinChIPseq( input, bins )
rep1.200bins = BinChIPseq( rep1, bins )
rep2.200bins = BinChIPseq( rep2, bins )

rep1.200bins

## GRanges object with 747586 ranges and 1 metadata column:
##             seqnames                 ranges strand   |     score
##                <Rle>              <IRanges>  <Rle>   | <integer>
##        [1]     chr6            [  1,  200]      *   |         0
##        [2]     chr6            [201,  400]      *   |         0
##        [3]     chr6            [401,  600]      *   |         0
##        [4]     chr6            [601,  800]      *   |         0
##        [5]     chr6            [801, 1000]      *   |         0
##        ...      ...                    ...    ... ...       ...
##   [747582]     chr6 [149516201, 149516400]      *   |         0
##   [747583]     chr6 [149516401, 149516600]      *   |         0
##   [747584]     chr6 [149516601, 149516800]      *   |         0
##   [747585]     chr6 [149516801, 149517000]      *   |         0
##   [747586]     chr6 [149517001, 149517037]      *   |         0
##   -------
##   seqinfo: 1 sequence from mm9 genome
```

We can quickly plot coverage for 1000 bins, starting from bin 200,000.

```
plot( 200000:201000, rep1.200bins$score[200000:201000],
   xlab="chr6", ylab="counts per bin" )
```



Below, we will see more sophisticaed ways of plotting coverage.

5.3 Exporting binned data

At this step of the analysis, the data is ready to be visualized and shared. One of the most common means of sharing ChIP-seq data is to generate .wig, .binWig or .bedGraph files. They are memory and size-efficient files holding the information about the signal along the genome. Moreover, these files can be visualized in genome browsers such as IGV and IGB. We show how to export the binned data as a .bedGraph file.

```
export(input.200bins,
       con='input_chr6.bedGraph',
       format = "bedGraph")
export(rep1.200bins,
       con='H3K27ac_rep1_chr6.bedGraph',
       format = "bedGraph")
export(rep2.200bins,
       con='H3K27ac_rep2_chr6.bedGraph',
       format = "bedGraph")
```

If you have a genome browser (like *IGB*) installed, have a look at the bedGraph files. In the next section, we see how to visualize them with *R*.

6 Visualisation of ChIP-seq data wit *Gviz*

Now, we have data which we would like to display along the genome. *R* offers a flexible infrastructure for visualisation of many types of genomics data. Here, we use the *Gviz* package for this purposes.

```
library(Gviz)
```

The principle of working with *Gviz* relies on the generation of tracks which can be, for example ChIP-seq signal along the genome, ChIP-seq peaks, gene models or any kind of other data such as annotation of CpG islands in the genome. We start with loading the gene models for chromosome 6 starting at position 122,530,000 and ending at position 122,900,000. We focus on this region as it harbors the *Nanog* gene, which is stongly expressed in ES cells.

We obtain the annotation using *biomaRt* package. Work with *biomaRt* package relies on querying the *biomart* database. In the *Appendix*, we show how to obtain gene models for protein coding genes for the archive mouse genome assembly (mm9) and how to generate the *bm* object holding the annotation of all the RefSeq genes.

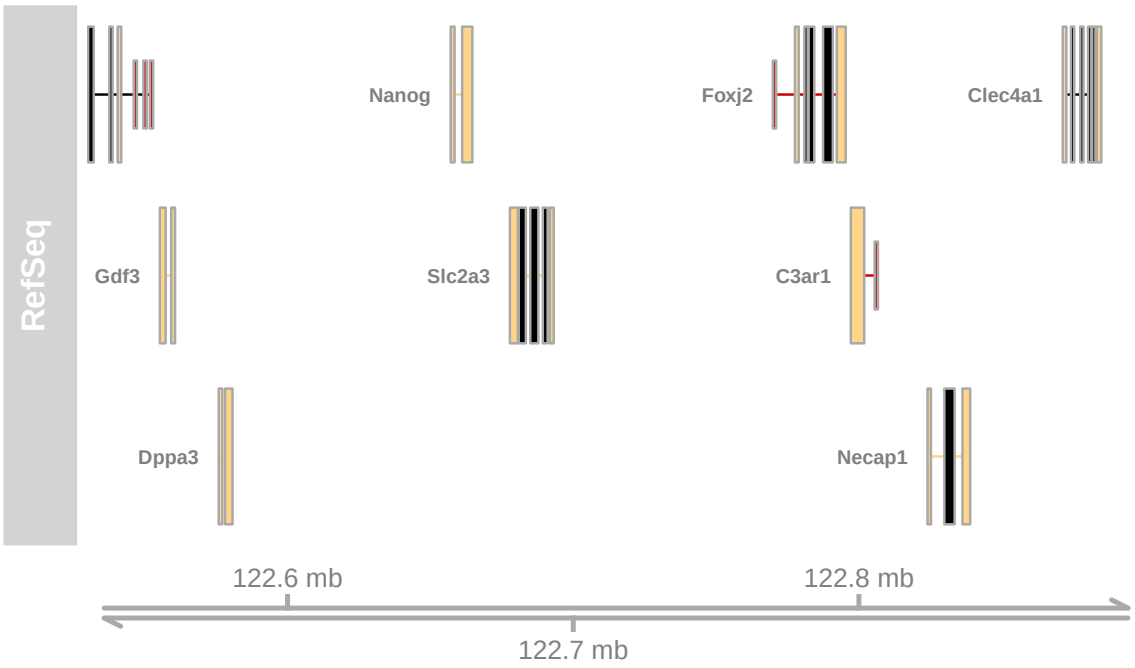
```
data(bm)
bm

## GeneRegionTrack 'RefSeq'
## | genome: mm9
## | active chromosome: chr6
## | annotation features: 102
```

We include the *GenomeAxisTrack* object which is a coordinate axis showing the genomic span of the analyzed region.

```
AT = GenomeAxisTrack( )
```

We plot the result using the *plotTracks* function. We choose the region to zoom into with the *from* and *to* arguments. The *transcriptAnnotation* argument allows to put the gene symbols in the plot.

```
plotTracks(c( bm, AT),
           from=122530000, to=122900000,
           transcriptAnnotation="symbol", window="auto",
           cex.title=1, fontsize=10 )
```

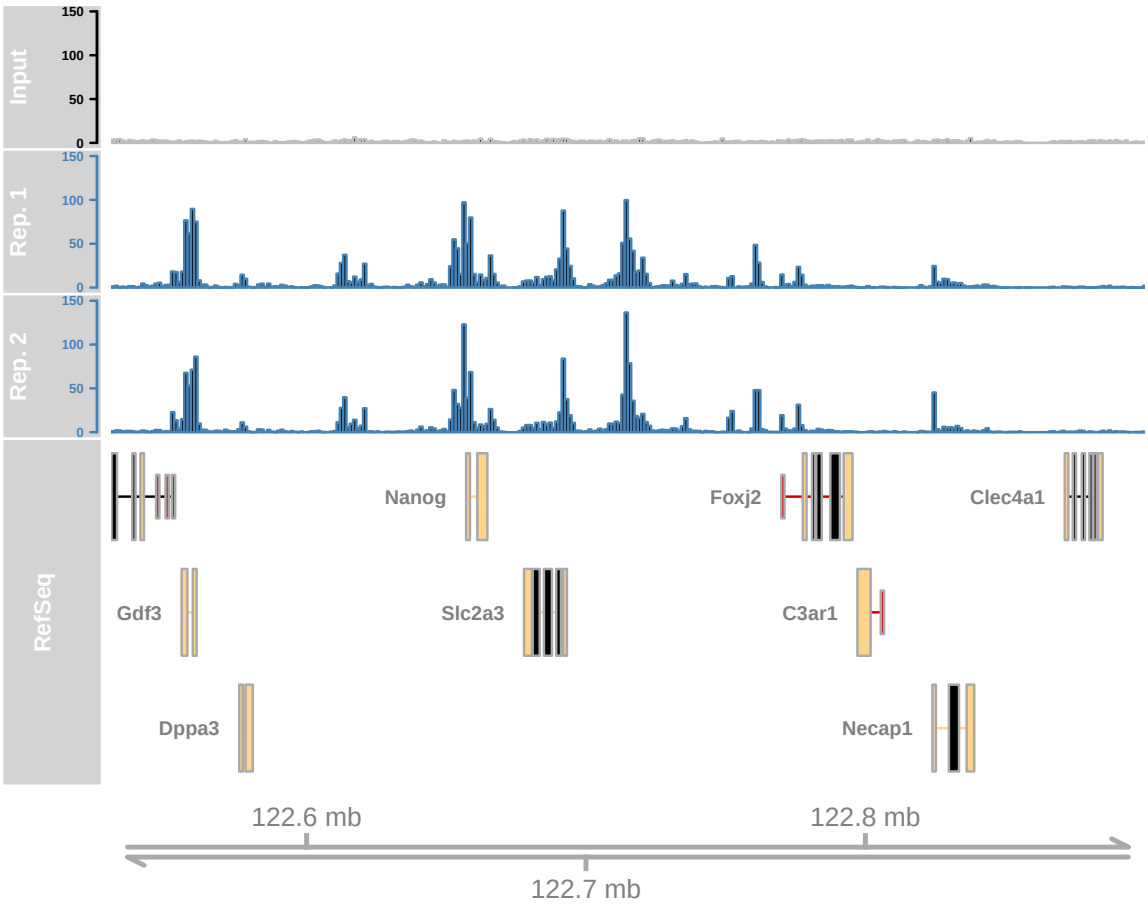


We next add our two data tracks to the plot. We first generate *DataTrack* objects with *DataTrack* function. We include the information about how the track is be labaled anc colored. We obtain *input.track*, *rep1.track* and *rep2.track* objects.

```
input.track = DataTrack(input.200bins,
                        strand="*", genome="mm9", col.histogram='gray',
                        fill.histogram='black', name="Input", col.axis="black",
                        cex.axis=0.4, ylim=c(0,150))

rep1.track = DataTrack(rep1.200bins,
                        strand="*", genome="mm9", col.histogram='steelblue',
                        fill.histogram='black', name="Rep. 1", col.axis='steelblue',
                        cex.axis=0.4, ylim=c(0,150))

rep2.track = DataTrack(rep2.200bins,
                        strand="*", genome="mm9", col.histogram='steelblue',
                        fill.histogram='black', name="Rep. 2", col.axis='steelblue',
                        cex.axis=0.4, ylim=c(0,150))
```

Finally, we plot these tracks along with the genomic features. We observe a uniform coverage in the case of the input track and pronounced peaks of enrichment H3K27ac in promoter and intergenic regions. Importantly, H3K27ac enriched regions are easily identified.

```
plotTracks(c(input.track, rep1.track, rep2.track, bm, AT),
           from=122530000, to=122900000,
           transcriptAnnotation="symbol", window="auto",
           type="histogram", cex.title=0.7, fontsize=10 )
```



7 ChIP-seq peaks

ChIP-seq experiments are designed to isolate regions enriched in a factor of interest. The identification of enriched regions, often refered to as peak finding, is an area of research by itself. There are many algorithms and tools used for peak finding. The choice of a method is strongly motivated by the kind of factor analyzed. For instance, transcription factor ChIP-seq yield well defined narrow peaks whereas histone modifications ChIP-seq experiments such as H3K36me3 yield extended regions of high coverage. Finally, ChIP-seq with antobodies recognizing polymerase II result in narrow peaks combined with extended regions of enrichment.

7.1 Identification of peaks

As we saw in the previous section of the tutorial, H3K27ac mark shows well defined peaks. In such a case, *MACS* is one of the most commonly used software for peak finding. ChIP-seq peak calling can also be done in *R* with the *BayesPeak* package. However, we stick here to the most common approach and use *MACS*. We ran *MACS* for you and provide the result in the data package. You can find the code necessary to obtain the peaks in the *Appendix* of the vignette.

7.2 Peaks – basic analysis in *R*

We next import the .bed files of the isolated peaks from the data package.

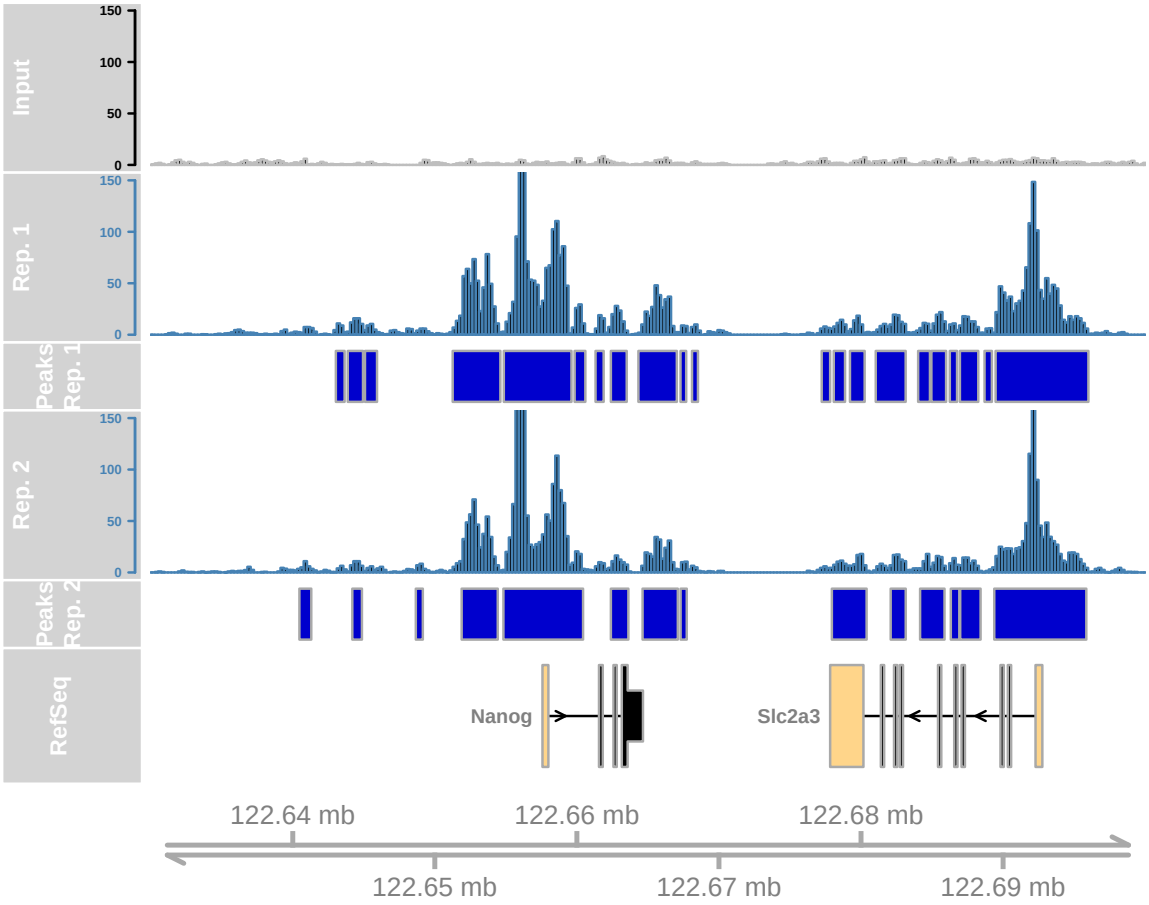
```
peaks.rep1 = import.bed(file.path(dataDirectory,'Rep1_peaks_ucsc_chr6.bed'),
                        asRangedData=FALSE)
peaks.rep2 = import.bed(file.path(dataDirectory,'Rep2_peaks_ucsc_chr6.bed'),
                        asRangedData=FALSE)
```

First step in the analysis of the identified peaks is to simply display them in the browser, along with the ChIP-seq and input tracks. To this end, we use *AnnotationTrack* function. We display peaks as boxes colored in blue.

```
peaks1.track = AnnotationTrack(peaks.rep1,
                               genome="mm9", name='Peaks Rep. 1',
                               chromosome='chr6',
                               shape='box',fill='blue3',size=2)
peaks2.track = AnnotationTrack(peaks.rep2,
                               genome="mm9", name='Peaks Rep. 2',
                               chromosome='chr6',
                               shape='box',fill='blue3',size=2)
```

We visualise the *Nanog* locus.

```
plotTracks(c(input.track, rep1.track, peaks1.track,
             rep2.track, peaks2.track, bm, AT),
           from=122630000, to=122700000,
           transcriptAnnotation="symbol", window="auto",
           type="histogram", cex.title=0.7, fontsize=10 )
```



We can see thath *MACS* has succesfully identified regions showing high H3K27ac signal. We see that both biological replicates agree well, however, in some cases peaks are called only in one sample. In the next section, we will analyse how often do we see the overlap between peaks and isolate reproducible peaks.

7.3 Venn diagrams

We first find the overlap between the peak sets of the two replicates.

```
ovlp = findOverlaps( peaks.rep1, peaks.rep2 )
ovlp
```

```
## Hits object with 2528 hits and 0 metadata columns:
##          queryHits subjectHits
##          <integer>   <integer>
##      [1]         1           1
##      [2]         2           2
##      [3]         3           3
##      [4]         4           4
##      [5]         5           5
##      ...       ...         ...
##   [2524]      3025        3025
##   [2525]      3026        3026
##   [2526]      3029        3027
```

```
##   [2527]      3030        3028
##   [2528]      3031        3030
##   -------
##   queryLength: 3032
##   subjectLength: 3032
```

If a peak in one replicate overlps with mutiple peaks in the other replicate, it will appear multiple times in *ovlp*. To see, how many peaks overlap with something in the other replicate, we count the number of unique peaks in each of the two columns of *ovlp* and take the smaller of these two counts to as the number of common peaks.

```
ov = min( length(unique(ovlp@queryHits)), length(unique(ovlp@subjectHits)) )
```

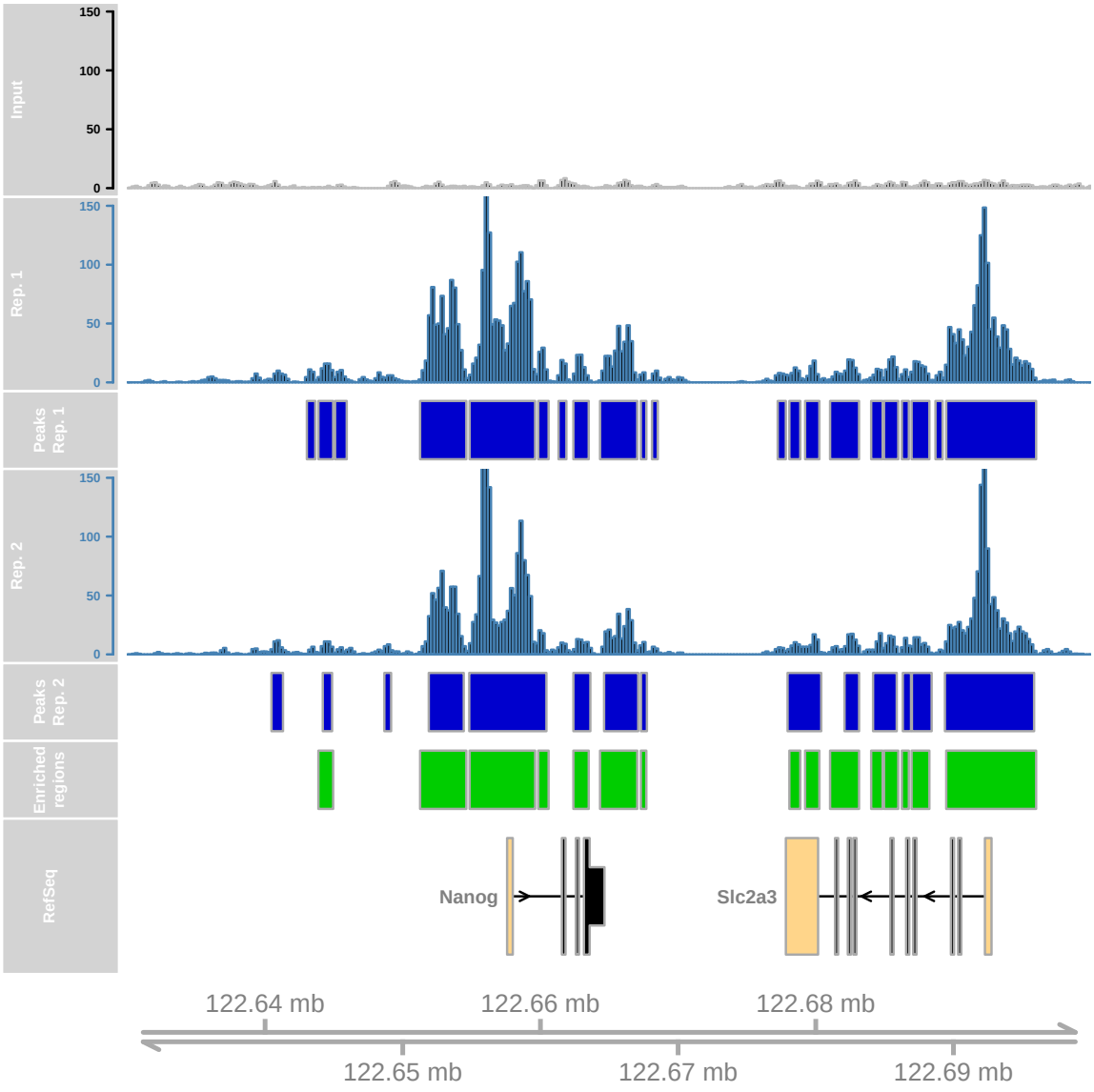
We draw this as a Venn diagram, using the *draw.pairwise.venn* function from the *VennDiagram* package.

```
library(VennDiagram)

draw.pairwise.venn(
   area1=length(peaks.rep1),
   area2=length(peaks.rep2),
   cross.area=ov,
   category=c("rep1", "rep2"),
   fill=c("steelblue", "blue3"),
   cat.cex=0.7)
```

```
## (polygon[GRID.polygon.549], polygon[GRID.polygon.550], polygon[GRID.polygon.551], polygon[GRID.polygo
```

We will focus only on peaks identified in both replicates (hereafter refered to as enriched areas). The enriched areas are colored in green.

```
enriched.regions = Reduce(subsetByOverlaps, list(peaks.rep1, peaks.rep2))

enr.reg.track = AnnotationTrack(enriched.regions,
                                genome="mm9", name='Enriched regions',
                                chromosome='chr6',
                                shape='box',fill='green3',size=2)

plotTracks(c(input.track, rep1.track, peaks1.track,
             rep2.track, peaks2.track, enr.reg.track,
             bm, AT),
           from=122630000, to=122700000,
           transcriptAnnotation="symbol", window="auto",
           type="histogram", cex.title=0.5, fontsize=10 )
```



7.4 Isolation of promoters overlapping H3K27ac peaks

One of the questions of a ChIP seq analyses is to which extend ChIP-enriched regions overlap a chosen type of features, such as promoters or regions enriched with other modifications. To this end, the overlap between peaks of ChIP-seq signal and the features of interest is analysed.

We exemplify such an analysis by testing how many of the H3K27ac enriched regions overlap promoter regions.

Identification of promoters

As shown in the Appendix, we have used *biomaRt* to get coordinates for start and end of all mouse genes. (These are the coordinates of the outermost UTR boundaries.) We load the results of the *biomaRt* query from the data package. It is given in the object *egs*, a *data.frame* containing *ensembl* ID along with gene symbols, genomic coordiantes and orientation of of mouse genes.

```
data(egs)
head(egs)
```

```
##      ensembl_gene_id external_gene_id chromosome_name start_position
## 1 ENSMUSG00000030270            Cpne9               6      113232301
## 2 ENSMUSG00000001632            Brpf1               6      113257175
## 3 ENSMUSG00000030271             Ogg1               6      113276966
## 4 ENSMUSG00000030272            Camk1               6      113284118
## 5 ENSMUSG00000048930            Tada3               6      113316019
## 6 ENSMUSG00000079426            Arpc4               6      113328107
##   end_position strand
## 1    113255621      1
## 2    113274851      1
## 3    113285062      1
## 4    113293978     -1
## 5    113327877     -1
## 6    113340442      1
```

We next identify the transcription start site (TSS), taking into account gene orientation.

```
egs$TSS = ifelse( egs$strand == "1", egs$start_position, egs$end_position )
head(egs)
```

```
##      ensembl_gene_id external_gene_id chromosome_name start_position
## 1 ENSMUSG00000030270            Cpne9               6      113232301
## 2 ENSMUSG00000001632            Brpf1               6      113257175
## 3 ENSMUSG00000030271             Ogg1               6      113276966
## 4 ENSMUSG00000030272            Camk1               6      113284118
## 5 ENSMUSG00000048930            Tada3               6      113316019
## 6 ENSMUSG00000079426            Arpc4               6      113328107
##   end_position strand       TSS
## 1    113255621      1 113232301
## 2    113274851      1 113257175
## 3    113285062      1 113276966
## 4    113293978     -1 113293978
## 5    113327877     -1 113327877
## 6    113340442      1 113328107
```

We consider regions of ± 200 bp around the TSS as promoters.

```
promoter_regions =
  GRanges(seqnames = Rle( paste0('chr', egs$chromosome_name) ),
          ranges = IRanges( start = egs$TSS - 200,
                            end = egs$TSS + 200 ),
          strand = Rle( rep("*", nrow(egs)) ),
          gene = egs$external_gene_id)
promoter_regions
```

```
## GRanges object with 1973 ranges and 1 metadata column:
##          seqnames               ranges strand   |        gene
##             <Rle>            <IRanges>  <Rle>   | <character>
##      [1]     chr6 [113232101, 113232501]      *   |       Cpne9
##      [2]     chr6 [113256975, 113257375]      *   |       Brpf1
##      [3]     chr6 [113276766, 113277166]      *   |        Ogg1
##      [4]     chr6 [113293778, 113294178]      *   |       Camk1
```

```
##      [5]     chr6 [113327677, 113328077]      *   |       Tada3
##      ...      ...                     ...    ... ...         ...
##   [1969]     chr6 [ 30119537,  30119937]      *   |      Mir183
##   [1970]     chr6 [ 93743566,  93743966]      *   |          U6
##   [1971]     chr6 [116321886, 116322286]      *   |     SNORA17
##   [1972]     chr6 [ 76150072,  76150472]      *   |          U1
##   [1973]     chr6 [106951246, 106951646]      *   |          U6
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Overlapping promoters with H3K27ac enriched regions

Now we would like to know how many of out the promoters overlap with a H3K27ac enriched regions.

```
ovlp2 = findOverlaps( enriched.regions, promoter_regions )

cat(sprintf( "%d of %d promoters are overlapped by an enriched region.",
   length( unique(ovlp2@subjectHits) ), length( promoter_regions ) ) )

## 634 of 1973 promoters are overlapped by an enriched region.
```

We can also turn the question around:

```
ovlp2b = findOverlaps( promoter_regions, enriched.regions )

cat(sprintf( "%d of %d enriched regions overlap a promoter.",
   length( unique(ovlp2b@subjectHits) ), length( enriched.regions ) ) )

## 546 of 2301 enriched regions overlap a promoter.
```

Is this a significant enrichment? To see, we first calculate how much chromosome 6 is part of a promotor region. The following command reduces the promotor list to non-overlapping intervals and sums up their widths

```
promotor_total_length = sum(width(reduce(promoter_regions)))
promotor_total_length

## [1] 766386
```

Which fraction of the chromsome is this?

```
promotor_fraction_of_chromosome_6 = promotor_total_length / seqlengths(si)["chr6"]
```

Nearly a quarter of promoters are overlapped by H3K27ac-enriched regions even though they make up only half a percent of the chromosome. Clearly, this is a strong enrichment. A binomial test confirms this:

```
binom.test( length( unique(ovlp2b@subjectHits) ), length( enriched.regions ), promotor_fraction_of_chror

##
##  Exact binomial test
##
## data:  length(unique(ovlp2b@subjectHits)) and length(enriched.regions)
## number of successes = 546, number of trials = 2301, p-value <
```

```
## 0.00000000000000022
## alternative hypothesis: true probability of success is not equal to 0.005125744
## 95 percent confidence interval:
##  0.2200317 0.2552159
## sample estimates:
## probability of success
##              0.2372881
```

Which promotors are overlapped with an H3K27ac peak? Let's see some examples:

```
pos.TSS = egs[ unique(findOverlaps( promoter_regions, enriched.regions )@queryHits),]
pos.TSS[1:3,]

##      ensembl_gene_id external_gene_id chromosome_name start_position
## 2 ENSMUSG00000001632            Brpf1               6      113257175
## 3 ENSMUSG00000030271             Ogg1               6      113276966
## 4 ENSMUSG00000030272            Camk1               6      113284118
##   end_position strand       TSS
## 2    113274851      1 113257175
## 3    113285062      1 113276966
## 4    113293978     -1 113293978
```

The first three promoters identified as overlapping a H3K27ac peak include: *Brpf1*, *Ogg1* and *Camk1 loci*.

7.5 Analysis of the distribution of H3K27ac around a subset of gene promoters

In this part of the analysis, we show how to generate plots displaying the distribution of ChIP-seq signal around certain genomic positions, here a set of promoter regions. These include a heatmap representation and an average profile for H3K27ac signal at promoters overlapping a peak of H3K27ac identified by *MACS*. These are one of the most frequently performed analysis steps in ChIP-seq experiments.

In the previous section, we have identified promoters overlaping a H3K27ac peak (the *pos.TSS* object). In order to get a comprehensive view of the distribution of H3K27ac around the corresponding TSS, we extend the analysed region to $\pm$1000 bp around the TSS. We divide each of these 2000 bp regions into 20 bins of 100 bp length each and order the bins with increasing position for genes on the '+' strand and decreasing for genes on the '-' strand.

Next, we tile the promoter regions with consecutive 100bp tiles. For each region, we order the tiles according to the gene orientation. We create 20 tiles per promoter region.

```
tiles = sapply( 1:nrow(pos.TSS), function(i)
   if( pos.TSS$strand[i] == "1" )
      pos.TSS$TSS[i] + seq( -1000, 900, length.out=20 )
   else
      pos.TSS$TSS[i] + seq( 900, -1000, length.out=20 ) )

tiles = GRanges(tilename = paste( rep( pos.TSS$ensembl_gene_id, each=20), 1:20, sep="_" ),
                seqnames = Rle( rep(paste0('chr', pos.TSS$chromosome_name), each=20) ),
                ranges = IRanges(start = as.vector(tiles),
                                 width = 100),
                strand = Rle(rep("*", length(as.vector(tiles)))),
                seqinfo=si)

tiles

## GRanges object with 12680 ranges and 1 metadata column:
```

```
##           seqnames                ranges strand   |              tilename
##              <Rle>             <IRanges>  <Rle>   |            <character>
##       [1]     chr6 [113256175, 113256274]      *   |  ENSMUSG00000001632_1
##       [2]     chr6 [113256275, 113256374]      *   |  ENSMUSG00000001632_2
##       [3]     chr6 [113256375, 113256474]      *   |  ENSMUSG00000001632_3
##       [4]     chr6 [113256475, 113256574]      *   |  ENSMUSG00000001632_4
##       [5]     chr6 [113256575, 113256674]      *   |  ENSMUSG00000001632_5
##       ...      ...                   ...    ... ...                    ...
##   [12676]     chr6   [30119137, 30119236]      *   | ENSMUSG00000065619_16
##   [12677]     chr6   [30119037, 30119136]      *   | ENSMUSG00000065619_17
##   [12678]     chr6   [30118937, 30119036]      *   | ENSMUSG00000065619_18
##   [12679]     chr6   [30118837, 30118936]      *   | ENSMUSG00000065619_19
##   [12680]     chr6   [30118737, 30118836]      *   | ENSMUSG00000065619_20
##   -------
##   seqinfo: 21 sequences from mm9 genome
```

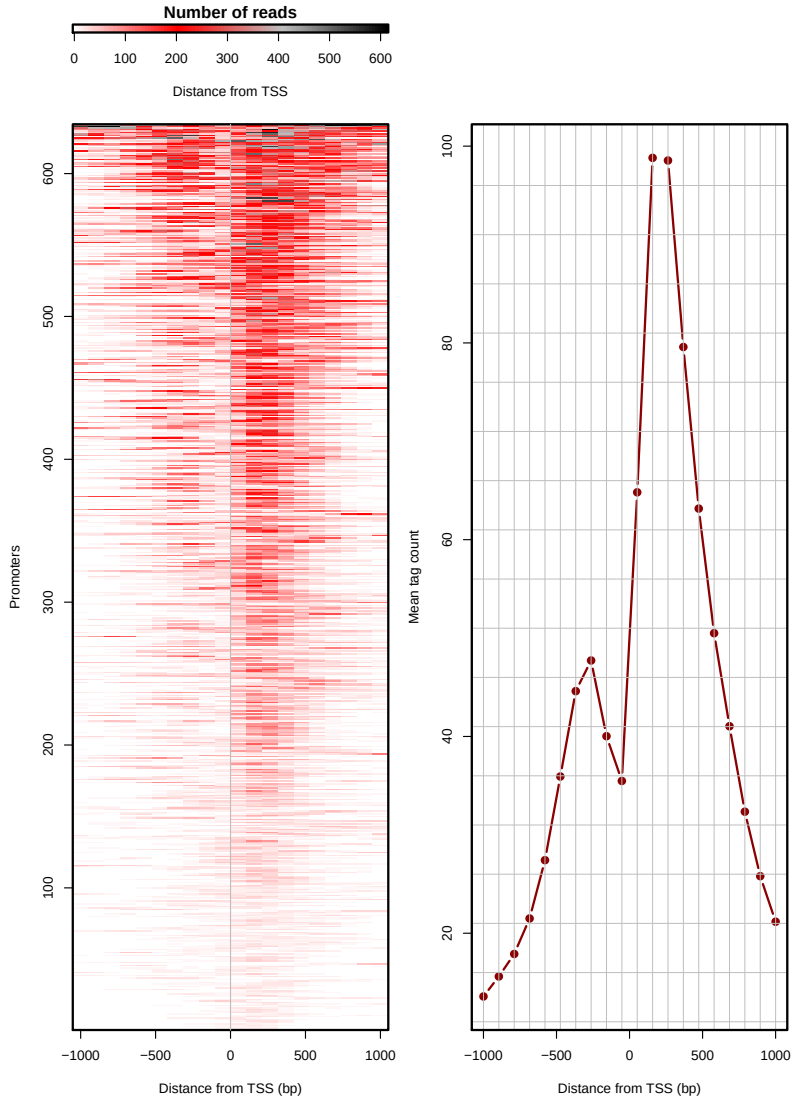
Next, we count how many reads are mapping to each tile. The resulting vector *H3K27ac.p* is next used to create a matrix (*H3K27ac.p.matrix*), where each row is a H3K27ac-enriched promoter. Each column coresponds to a consecutive 100bp tile of 2000 bp region around the TSS overlapping a H3K27ac peak. Since we have divided each promoter region in 21 tiles, we obtain a matrix with 21 columns and 634 rows (the number of promoters overlapping H3K27ac peak).

```
H3K27ac.p = countOverlaps( tiles, rep1) +
  countOverlaps( tiles, rep2 )

H3K27ac.p.matrix = matrix( H3K27ac.p, nrow=nrow(pos.TSS),
                           ncol=20, byrow=TRUE )
```

Finally, we plot the result as a heatmap and as a plot of average values per each tile for all the included promoters.

```
colors = colorRampPalette(c('white','red','gray','black'))(100)

layout(mat=matrix(c(1,2,0,3), 2, 2),
       widths=c(2,2,2),
       heights=c(0.5,5,0.5,5), TRUE)


par(mar=c(4,4,1.5,1))
image(seq(0, max(H3K27ac.p.matrix), length.out=100), 1,
      matrix(seq(0, max(H3K27ac.p.matrix), length.out=100),100,1),
      col = colors,
      xlab='Distance from TSS', ylab='',
      main='Number of reads', yaxt='n',
      lwd=3, axes=TRUE)
box(col='black', lwd=2)
image(x=seq(-1000, 1000, length.out=20),
      y=1:nrow(H3K27ac.p.matrix),
      z=t(H3K27ac.p.matrix[order(rowSums(H3K27ac.p.matrix)),]),
      col=colors,
      xlab='Distance from TSS (bp)',
      ylab='Promoters', lwd=2)
box(col='black', lwd=2)
abline(v=0, lwd=1, col='gray')
```

```
plot(x=seq(-1000, 1000, length.out=20),
     y=colMeans(H3K27ac.p.matrix),
     ty='b', pch=19,
     col='red4',lwd=2,
     ylab='Mean tag count',
     xlab='Distance from TSS (bp)')
abline(h=seq(1,100,by=5),
       v=seq(-1000, 1000, length.out=20),
       lwd=0.25, col='gray')
box(col='black', lwd=2)
```



We observe a strong enrichment of H3K27ac modification right after the TSS and a weaker peak of H3K27ac at the region immediately upstream of the TSS.

8 Session info

```
sessionInfo()
```

```
## R version 3.2.0 (2015-04-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.2 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] grid      stats4    parallel  stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] VennDiagram_1.6.9          Gviz_1.12.0
##  [3] chipseq_1.18.0             ShortRead_1.26.0
##  [5] GenomicAlignments_1.4.1    Rsamtools_1.20.4
##  [7] BiocParallel_1.2.3         BSgenome_1.36.0
##  [9] Biostrings_2.36.1          XVector_0.8.0
## [11] rtracklayer_1.28.4         GenomicRanges_1.20.5
## [13] GenomeInfoDb_1.4.0         IRanges_2.2.4
## [15] S4Vectors_0.6.0            BiocGenerics_0.14.0
## [17] EpigeneticsCSAMA2015_0.0.2 knitr_1.10.5
##
## loaded via a namespace (and not attached):
##  [1] VariantAnnotation_1.14.3 reshape2_1.4.1
##  [3] splines_3.2.0            lattice_0.20-31
##  [5] colorspace_1.2-6         GenomicFeatures_1.20.1
##  [7] XML_3.98-1.2             survival_2.38-2
##  [9] foreign_0.8-63           DBI_0.3.1
## [11] RColorBrewer_1.1-2       lambda.r_1.1.7
## [13] matrixStats_0.14.0       plyr_1.8.3
## [15] stringr_1.0.0            zlibbioc_1.14.0
## [17] munsell_0.4.2            gtable_0.1.2
## [19] futile.logger_1.4.1      hwriter_1.3.2
## [21] evaluate_0.7             latticeExtra_0.6-26
## [23] Biobase_2.28.0           biomaRt_2.24.0
## [25] AnnotationDbi_1.30.1     highr_0.5
## [27] proto_0.3-10             Rcpp_0.11.6
## [29] acepack_1.3-3.3          scales_0.2.5
## [31] formatR_1.2              Hmisc_3.16-0
## [33] gridExtra_0.9.1          ggplot2_1.0.1
## [35] digest_0.6.8             stringi_0.4-1
## [37] biovizBase_1.16.0        tools_3.2.0
## [39] bitops_1.0-6             magrittr_1.5
## [41] RCurl_1.95-4.6           RSQLite_1.0.0
## [43] dichromat_2.0-0          Formula_1.2-1
## [45] cluster_2.0.1            futile.options_1.0.0
## [47] MASS_7.3-40              rpart_4.1-9
```

```
## [49] nnet_7.3-9
```

9 Appendix

9.1 Obtaining data from European Nucleotide Archive

The European Nucleotide Archive (http://www.ebi.ac.uk/ena) provides many types of raw sequencing data, sequence assembly information and functional annotation. We download the data corresponding to ChIP-seq experiment mapping the H3K27ac histone modification in mouse Embryonic Stem cells (mES cells) along with the input control sample from the study *Histone H3K27ac separates active from poised enhancers and predicts developmental state* by Creyghton *et al*.

```
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR066/SRR066787/SRR066787.fastq.gz .
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR066/SRR066766/SRR066766.fastq.gz .
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR066/SRR066767/SRR066767.fastq.gz .
```

9.2 Read quality

Read quality is the first step in all the analyses of sequenced reads. The package *ShortRead* provides a function taking as input the .fastq files downloaded from the ENA database. We first generate a vector with fastq file names.

```
fls = list.files(dataDirectory, ".fastq$", full=TRUE)
names(fls) = sub(".fastq", "", basename(fls))
```

We read each of these files and apply the *qas* function assessing the quality of the reads in each file. Finally, we generate a *HTML* quality report.

```
library(ShortRead)
qas = lapply(seq_along(fls),
             function(i, fls) qa(readFastq(fls[i]), names(fls)[i]),
             fls)
qa = do.call(rbind, qas)
rpt = report(qa,dest = 'QA_report.html')
```

9.3 External file preparations

The next step is to align the reads to mm9 mouse genome assembly. This is done using *Bowtie2* tool. The resulting .sam files are next transformed to .bam files and filtered for best aligned reads using *samtools*. PCR duplicates are removed. BAM files are next transfomed to bed files. For the sake of consistency with other tools, in the final step of data preprocessing we add a 'chr' prefix to the chromosome names using *awk*.

```
gunzip SRR066787.fastq.gz
gunzip SRR066766.fastq.gz
gunzip SRR066767.fastq.gz
```

9.4 Alignment

```
bowtie2 -p 8 -q NCBIM37.67 SRR066787.fastq -S ES_input.sam
bowtie2 -p 8 -q NCBIM37.67 SRR066766.fastq -S H3K27ac_rep1.sam
bowtie2 -p 8 -q NCBIM37.67 SRR066767.fastq -S H3K27ac_rep2.sam
```

9.5 Retaining only best alignments

```
samtools view -bS -q 40 ES_input.sam > ES_input_bestAlignment.bam
samtools view -bS -q 40 H3K27ac_rep1.sam > H3K27ac_rep1_bestAlignment.bam
samtools view -bS -q 40 H3K27ac_rep2.sam > H3K27ac_rep2_bestAlignment.bam
```

9.6 PCR duplicate removal

```
samtools rmdup -s  ES_input_bestAlignment.bam ES_input_filtered.bam
samtools rmdup -s  H3K27ac_rep1_bestAlignment.bam H3K27ac_rep1_filtered.bam
samtools rmdup -s  H3K27ac_rep2_bestAlignment.bam H3K27ac_rep2_filtered.bam
```

9.7 Transforming reads to .bed format

```
bedtools bamtobed -i ES_input_filtered.bam > ES_input_filtered.bed
bedtools bamtobed -i H3K27ac_rep1_filtered.bam > H3K27ac_rep1_filtered.bed
bedtools bamtobed -i H3K27ac_rep2_filtered.bam > H3K27ac_rep2_filtered.bed
```

9.8 Additional preparations

```
awk '$0="chr"$0' ES_input_filtered.bed > ES_input_filtered_ucsc.bed
awk '$0="chr"$0' H3K27ac_rep1_filtered.bed > H3K27ac_rep1_filtered_ucsc.bed
awk '$0="chr"$0' H3K27ac_rep2_filtered.bed > H3K27ac_rep2_filtered_ucsc.bed
```

Finally, for the purpose of this lab, we isolate data for only one chromosome (chr6).

```
awk '{if($1=="chr6") print $0}' ES_input_filtered_ucsc.bed
> ES_input_filtered_ucsc_chr6.bed
awk '{if($1=="chr6") print $0}' H3K27ac_rep1_filtered_ucsc.bed
> H3K27ac_rep1_filtered_ucsc_chr6.bed
awk '{if($1=="chr6") print $0}' H3K27ac_rep2_filtered_ucsc.bed
> H3K27ac_rep2_filtered_ucsc_chr6.bed
```

Obtaining object *si* for *mm9*

We obtain chromosome lengths from the *BSgenome.Mmusculus.UCSC.mm9* package. The chromosome names in the *si* file are in the *ensembl* format, we add a prefix 'chr' to chromosome names.

```
library(BSgenome.Mmusculus.UCSC.mm9)
genome = BSgenome.Mmusculus.UCSC.mm9
si = seqinfo(genome)
si = si[ paste0('chr', c(1:19, 'X', 'Y'))]
```

Obtaining object *bm* for *mm9*

```
library(biomaRt)
mart = useMart(biomart = "ENSEMBL_MART_ENSEMBL",
               dataset = "mmusculus_gene_ensembl",
               host="may2012.archive.ensembl.org")
fm = Gviz:::.getBMFeatureMap()
fm["symbol"] = "external_gene_id"
```

Next, we get a snapshot of the results for chromosome 6 starting at position 122530000 and ending at position 122900000. This region amongst others encodes a highly ES cell specific *Nanog* gene. We first isolate gene models for this interval. The result *bm* is saved in the data directory.

```
bm = BiomartGeneRegionTrack(chromosome='chr6', genome="mm9",
                            start=122530000, end = 122900000,
                            biomart=mart,filter=list("with_ox_refseq_mrna"=TRUE),
                            size=4, name="RefSeq", utr5="red3", utr3="red3",
                            protein_coding="black", col.line=NULL, cex=7,
                            collapseTranscripts="longest",
                            featureMap=fm)
```

Peak finding with *MACS*

```
macs14 -t H3K27ac_rep1_filtered.bed -c ES_input_filtered_ucsc.bed -f BED -g mm --nomodel -n Rep1
macs14 -t H3K27ac_rep2_filtered.bed -c ES_input_filtered_ucsc.bed -f BED -g mm --nomodel -n Rep2
awk '$0="chr"$0' Rep1_peaks.bed > Rep1_peaks_ucsc.bed
awk '$0="chr"$0' Rep2_peaks.bed > Rep2_peaks_ucsc.bed
awk '{if($1=="chr6") print $0}' Rep1_peaks_ucsc.bed > Rep1_peaks_ucsc_chr6.bed
awk '{if($1=="chr6") print $0}' Rep2_peaks_ucsc.bed > Rep2_peaks_ucsc_chr6.bed
```

Promoter isolation

Here we provide the code necessary to isolate gene models from the *biomart* data base. The object *egs* contains the annotation of the most external 5 and 3 prime UTRs for each gene model.

```
listAttributes(mart)[1:3,]
ds = useDataset('mmusculus_gene_ensembl', mart=mart)
chroms = 6

egs = getBM(attributes = c('ensembl_gene_id','external_gene_id',
                           'chromosome_name','start_position',
                           'end_position','strand'),
            filters='chromosome_name',
            values=chroms,
            mart=ds)
```