Some Basic ChIP-Seq Data Analysis

July 28, 2009

Our goal is to describe the use of Bioconductor software to perform some basic tasks in the analysis of ChIP-Seq data. We will use several functions in the chipseq package, which provides convenient interfaces to other powerful packages such as ShortRead and IRanges. We will also use the lattice package for visualization.

```
> library(chipseq)
> library(GenomicFeatures)
> library(lattice)
```

Example data

The `cstest` data set is included in the chipseq package. The dataset contains data for three chromosomes from Solexa lanes, one from a CTCF mouse ChIP-Seq, and one from a GFP mouse ChIP-Seq. The raw reads were aligned to the reference genome (mouse in this case) using an external program (MAQ), and the results read in using the `readReads` function, which in turn uses the `readAligned` function in the ShortRead package. This step removed all duplicate reads and applied a quality score cutoff. The remaining data were reduced to a set of alignment start positions (including orientation).

```
> data(cstest)
> cstest

  A GenomeDataList instance of length 2
```

`cstest` is an object of class *"GenomeDataList"*, and has a list-like structure, each component a *"GenomeData"* object representing data from one lane.

```
> cstest$ctcf

A GenomeData instance for Mmusculus
chromosomes(3): chr10 chr11 chr12
```

Each of these are themselves lists containing positive and negative strand alignments:

```
> str(cstest$ctcf$chr10)

List of 2
 $ -: int [1:72371] 3012999 3013096 3013098 3013135 3032735 3040511 3040520 3041297 3044041 304504
 $ +: int [1:73179] 3012936 3012941 3012944 3012955 3012963 3012969 3012978 3013071 3018464 302076
```

The aligned position of the first cycle is stored.

The mouse genome

The data we have refer to alignments to a genome, and only makes sense in that context. Bioconductor has genome packages containing the full sequences of several genomes. The one relevant for us is

```
> library(BSgenome.Mmusculus.UCSC.mm9)
> mouse.chromlens <- seqlengths(Mmusculus)
> head(mouse.chromlens)
```

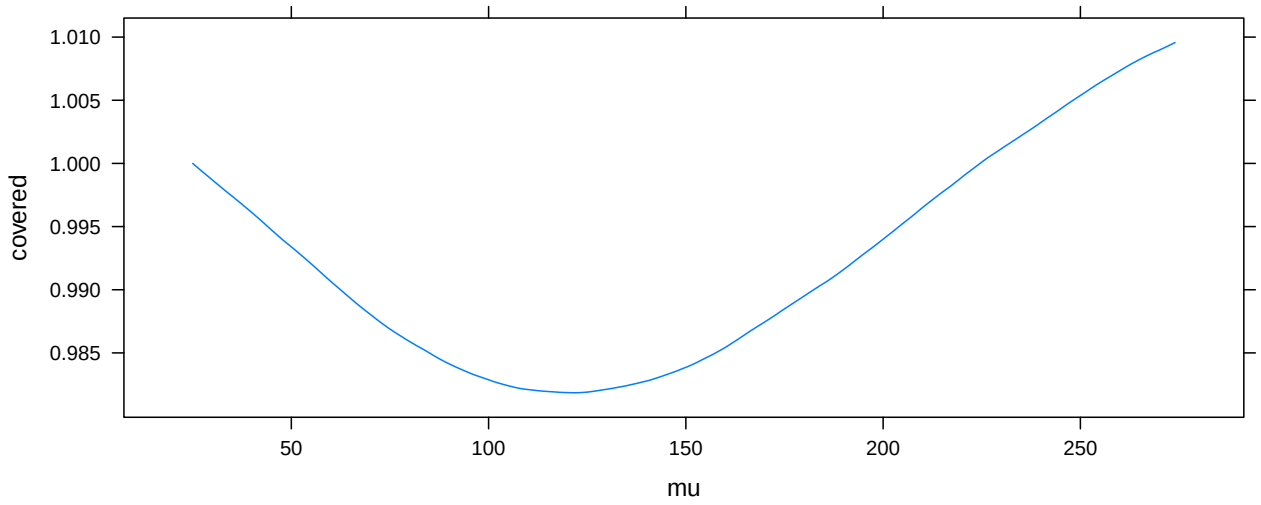
```
     chr1      chr2      chr3      chr4      chr5      chr6
197195432 181748087 159599783 155630120 152537259 149517037
```

We will only make use of the chromosome lengths, but the actual sequence will be needed for motif finding, etc.

Extending reads

Solexa gives us the first few (24 in this example) bases of each fragment it sequences, but the actual fragment is longer. By design, the sites of interest (transcription factor binding sites) should be somewhere in the fragment, but not necessarily in its initial part. Although the actual lengths of fragments vary, extending the alignment of the short read by a fixed amount in the appropriate direction, depending on whether the alignment was to the positive or negative strand, makes it more likely that we cover the actual site of interest.
How much should we extend? A simple choice is some estimate of the average fragment length. The following plots the number of bases covered by unextended reads after the negative strand reads are shifted, with varying amounts of shift associated with varying average fragment length. There is a clear minima, which can be used as an estimate of the average fragment length.

```
> bc <- basesCovered(cstest$ctcf$chr10, shift = 1:250, seqLen = 24)
> xyplot(covered ~ mu, bc, type = "l")
```



Exercise 1
What would you expect the plot to be like in control data with no signal? Create a similar plot for `cstest$gfp$chr10`.

We extend all reads to be 150 bases long. This is done using the `extendReads()` function, which works on data from one chromosome in one lane.

```
> ext <- extendReads(cstest$ctcf$chr10, seqLen = 150)
> head(ext)

IRanges instance:
      start     end width
[1] 3012936 3013085   150
[2] 3012941 3013090   150
[3] 3012944 3013093   150
[4] 3012955 3013104   150
[5] 3012963 3013112   150
[6] 3012969 3013118   150
```

The result is essentially a collection of intervals (ranges) over the reference genome. Most subsequent steps will work on the results of this step. We can convert all our data to this form as a preliminary step. However, the extension step is reasonably fast, and it is often more memory-efficient to perform the extension on-the-fly whenever necessary.

Coverage, islands, and depth

A useful summary of this information is the *coverage*, that is, how many times each base in the genome was covered by one of these intervals.

```
> cov <- coverage(ext, width = mouse.chromlens["chr10"])
> cov

  'integer' Rle instance of length 129993255 with 288055 runs
  Lengths:  3012849 86 5 3 3 2 6 8 6 9 ...
  Values :  0 1 2 3 4 5 6 7 8 9 ...
```

For efficiency, the result is stored in a run-length encoded form.

The regions of interest are contiguous segments of non-zero coverage, also known as *islands*.

```
> islands <- slice(cov, lower = 1)
> islands
```

```
  Views on a 129993255-length Rle subject

views:
             start       end width
    [1]   3012850   3013220   371 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [2]   3018464   3018613   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [3]   3020766   3020915   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [4]   3023019   3023168   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [5]   3023240   3023389   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [6]   3032586   3032735   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [7]   3038377   3038526   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [8]   3040355   3040520   166 [1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 ...]
    [9]   3041148   3041297   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    ...       ...       ...   ... ...
[93350] 129973225 129973447   223 [1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...]
[93351] 129974863 129975012   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93352] 129975575 129975724   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93353] 129978669 129978818   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93354] 129979259 129979521   263 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93355] 129980303 129980452   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93356] 129981957 129982106   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93357] 129982380 129982529   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[93358] 129987020 129987169   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
```

For each island, we can compute the number of reads in the island, and the maximum coverage depth within that island.

```
> viewSums(head(islands))
```

```
[1] 1800  150  150  150  150  150
```

```
> viewMaxs(head(islands))
```

```
[1] 11  1  1  1  1  1
```

```
> nread.tab <- table(viewSums(islands) / 150)
> depth.tab <- table(viewMaxs(islands))
> head(nread.tab, 10)
```

```
    1     2     3     4     5     6     7     8     9    10
75820 12146  2368   679   320   198   171   109   125   107
```

```
> head(depth.tab, 10)
```

```
    1     2     3     4     5     6     7     8     9    10
75873 13170  1824   422   255   158   158   125   119   111
```

Exercise 2

Repeat these steps for the `gfp` *dataset.*

Processing multiple lanes

Although data from one chromosome within one lane is often the natural unit to work with, we typically want to apply any procedure to all chromosomes in all lanes. A function that is useful for this purpose is `gdapply`, which recursively applies a summary function to a full dataset. If the summary function produces a data frame, the result can be coerced into a flat data frame that is often easier to work with. Here is a simple summary function that computes the frequency distribution of the number of reads.

```
> islandReadSummary <- function(x)
+ {
+     g <- extendReads(x, seqLen = 150)
+     s <- slice(coverage(g), lower = 1)
+     tab <- table(viewSums(s) / 150)
+     ans <- data.frame(nread = as.numeric(names(tab)), count = as.numeric(tab))
+     ans
+ }
```

Applying it to our test-case, we get

```
> head(islandReadSummary(cstest$ctcf$chr10))
```

```
  nread count
1     1 75820
2     2 12146
3     3  2368
4     4   679
5     5   320
6     6   198
```

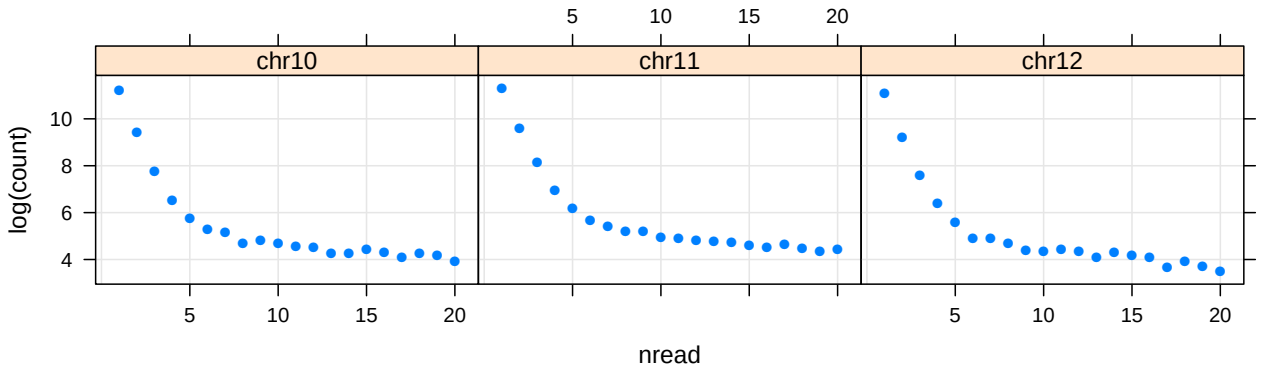
We can now use it to summarize the full dataset.

```
> nread.islands <- gdapply(cstest, islandReadSummary)
> nread.islands <- as(nread.islands, "data.frame")
> head(nread.islands)
```

```
  nread count chromosome sample
1     1 75820      chr10   ctcf
2     2 12146      chr10   ctcf
3     3  2368      chr10   ctcf
4     4   679      chr10   ctcf
5     5   320      chr10   ctcf
6     6   198      chr10   ctcf
```

It is now easy to plot the results.

```
> xyplot(log(count) ~ nread | chromosome, nread.islands,
+        subset = (sample == "ctcf" & nread <= 20),
+        layout = c(3, 1), pch = 16, type = c("p", "g"))
```

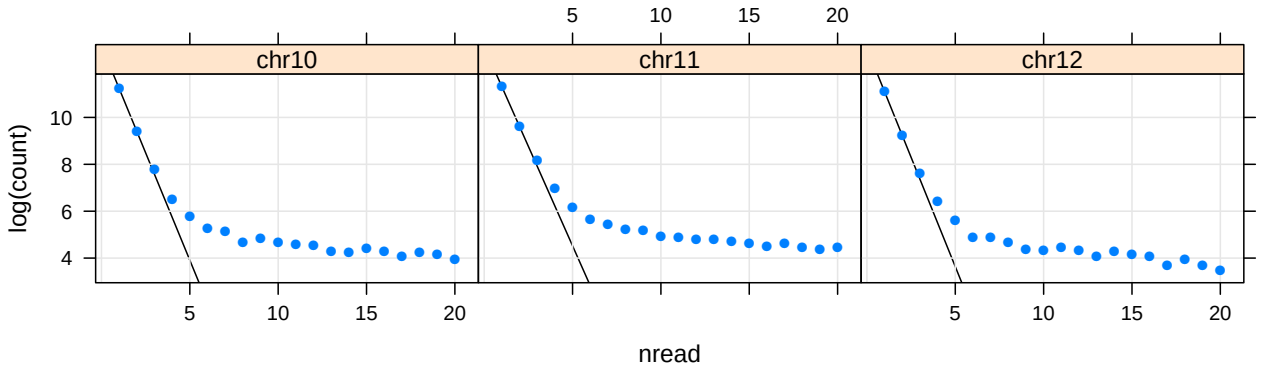


If reads were sampled randomly from the genome, then the null distribution of the number of reads per island would have a geometric distribution; that is,

$$P(X = k) = p^{k-1}(1 - p)$$

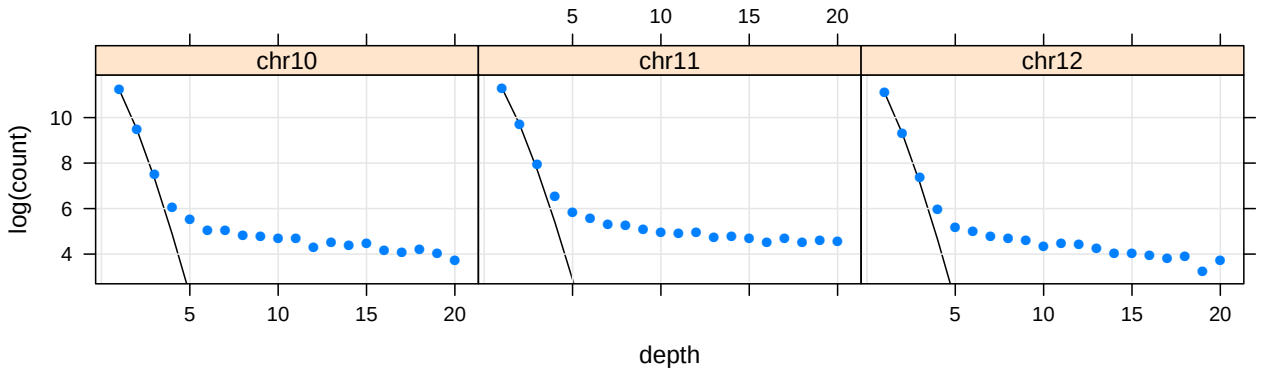
In other words, $\log P(X = k)$ is linear in k. Although our samples are not random, the islands with just one or two reads may be representative of the background, and can be used to estimate the parameter of the null distribution. Graphically, this is the line passing through the first two points.

```
> xyplot(log(count) ~ nread | chromosome, nread.islands,
+        subset = (sample == "ctcf" & nread <= 20),
+        layout = c(3, 1), pch = 16, type = c("p", "g"),
+        panel = function(x, y, ...) {
+            panel.lmline(x[1:2], y[1:2], col = "black")
+            panel.xyplot(x, y, ...)
+        })
```



We can create a similar plot of the distribution of depths, this time fitting a Poisson distribution through the first two points.

```
> islandDepthSummary <- function(x)
+ {
+     g <- extendReads(x, seqLen = 150)
+     s <- slice(coverage(g), lower = 1)
+     tab <- table(viewMaxs(s))
+     ans <- data.frame(depth = as.numeric(names(tab)), count = as.numeric(tab))
+     ans
+ }
> depth.islands <- gdapply(cstest, islandDepthSummary)
> depth.islands <- as(depth.islands, "data.frame")
> xyplot(log(count) ~ depth | chromosome, depth.islands,
+        subset = (sample == "ctcf" & depth <= 20),
+        layout = c(3, 1), pch = 16, type = c("p", "g"),
+        panel = function(x, y, ...) {
+            lambda <- 2 * exp(y[2]) / exp(y[1])
+            null.est <- function(xx) {
+                xx * log(lambda) - lambda - lgamma(xx + 1)
+            }
+            log.N.hat <- null.est(1) - y[1]
+            panel.lines(1:10, -log.N.hat + null.est(1:10), col = "black")
+            panel.xyplot(x, y, ...)
+        })
```



Exercise 3

Produce similar plots for the `gfp` *dataset. What qualitatitve differences do you see? What would be a reasonbale cutoff for deciding that the depth of an island is too high to be explained by chance, and hence is likely to contain a CTCF binding site?*

Peaks

Going back to our example of chr10 of the first sample, we can define "peaks" to be regions of the genome where coverage is 8 or more.

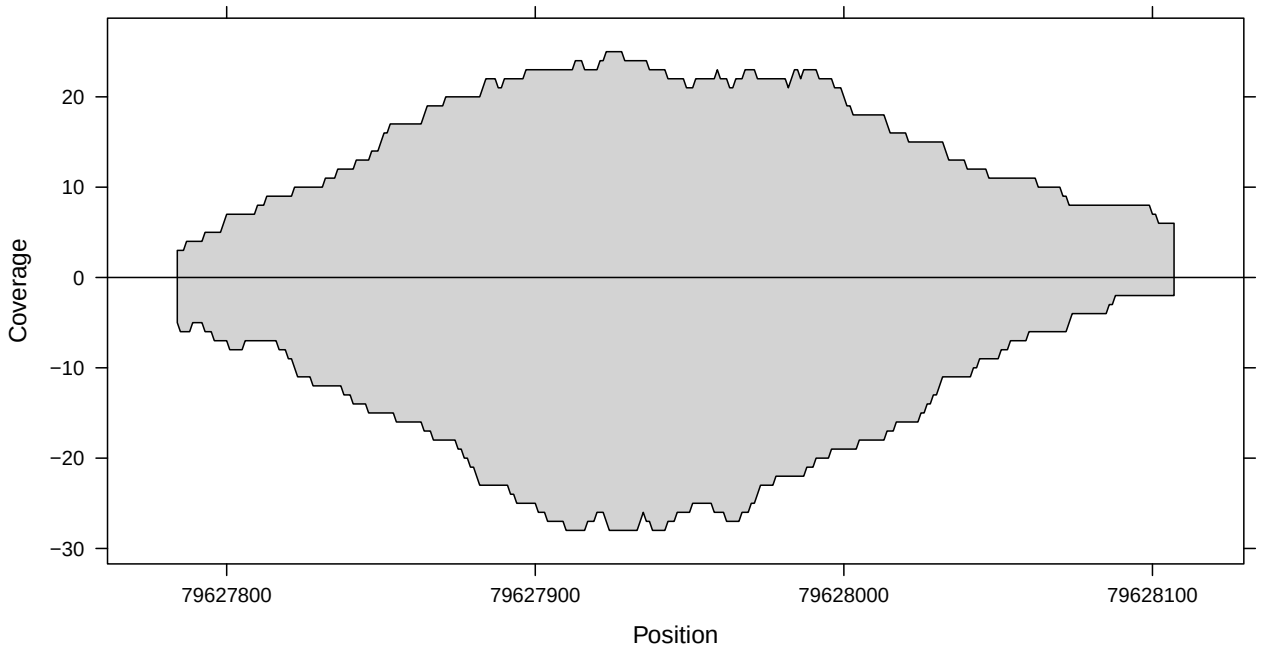
```
> peaks <- slice(cov, lower = 8)
> peaks

  Views on a 129993255-length Rle subject

views:
           start       end width
   [1]   3012963   3013096   134 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
   [2]   3234799   3234891    93 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [3]   3270074   3270078     5 [8 8 8 8 8]
   [4]   3270088   3270260   173 [ 8  8  8  8  8  8  9 10 11 11 12 12 12 ...]
   [5]   3277695   3277811   117 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [6]   3460866   3460928    63 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [7]   3617850   3617947    98 [8 8 8 8 8 8 8 9 9 9 8 9 9 9 9 9 9 9 9 9 ...]
   [8]   3651762   3651957   196 [ 8  8 10 10 10 12 12 12 12 12 12 12 12 ...]
   [9]   4310429   4310670   242 [ 8  9  9 10 10 10 11 11 11 11 11 11 12 ...]
   ...       ...       ...   ... ...
[1669] 128463192 128463367   176 [ 8  8  9  9  9  9  9  9  9  9  9  9  9 ...]
[1670] 128986519 128986595    77 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1671] 128986604 128986610     7 [8 8 8 8 8 8 8]
[1672] 129058941 129058952    12 [8 8 8 8 8 8 8 8 8 8 8 8]
[1673] 129530064 129530177   114 [ 9  9  9  9  9  9  9  9  9  9  9  9  9 ...]
[1674] 129533331 129533381    51 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1675] 129665395 129665570   176 [ 8  9  9 10 10 10 11 12 12 12 12 12 12 ...]
[1676] 129666830 129666897    68 [8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 ...]
[1677] 129750671 129750808   138 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
```

It is meaningful to ask about the contribution of each strand to each peak, as the sequenced region of pull-down fragments would be on opposite sides of a binding site depending on which strand it matched. We can compute strand-specific coverage, and look at the individual coverages under the combined peaks as follows:

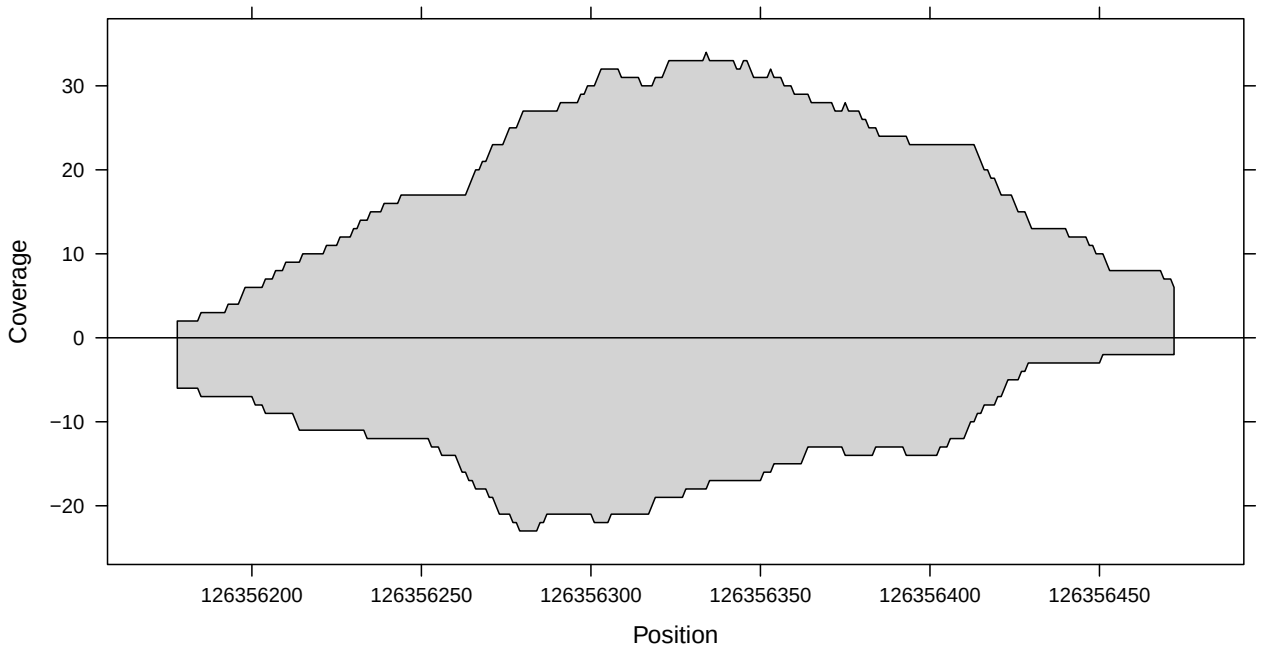
```
> peak.depths <- viewMaxs(peaks)
> cov.pos <- coverage(extendReads(cstest$ctcf$chr10, strand = "+", seqLen = 150),
+                     width = mouse.chromlens["chr10"])
> cov.neg <- coverage(extendReads(cstest$ctcf$chr10, strand = "-", seqLen = 150),
+                     width = mouse.chromlens["chr10"])
> peaks.pos <- copyIRanges(peaks, cov.pos)
> peaks.neg <- copyIRanges(peaks, cov.neg)
> wpeaks <- tail(order(peak.depths), 4)
> wpeaks

[1]  942 1558  875 1025
```

We plot four highest peaks below.

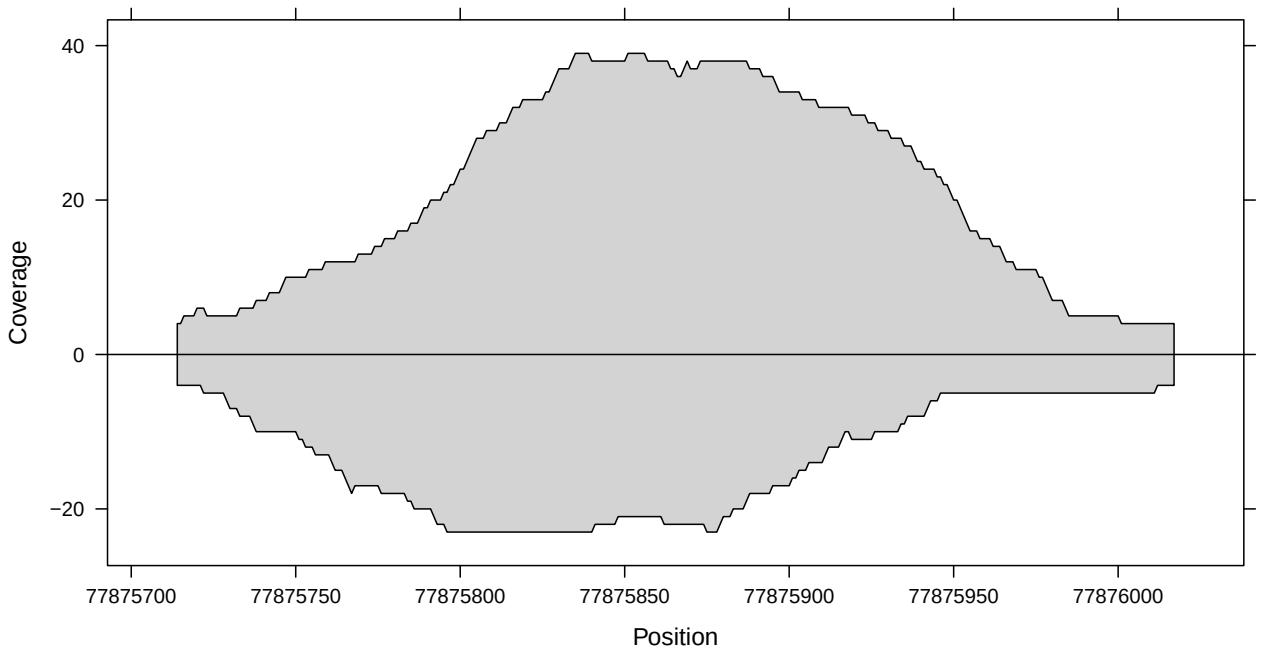
```
> coverageplot(peaks.pos[wpeaks[1]], peaks.neg[wpeaks[1]])
```



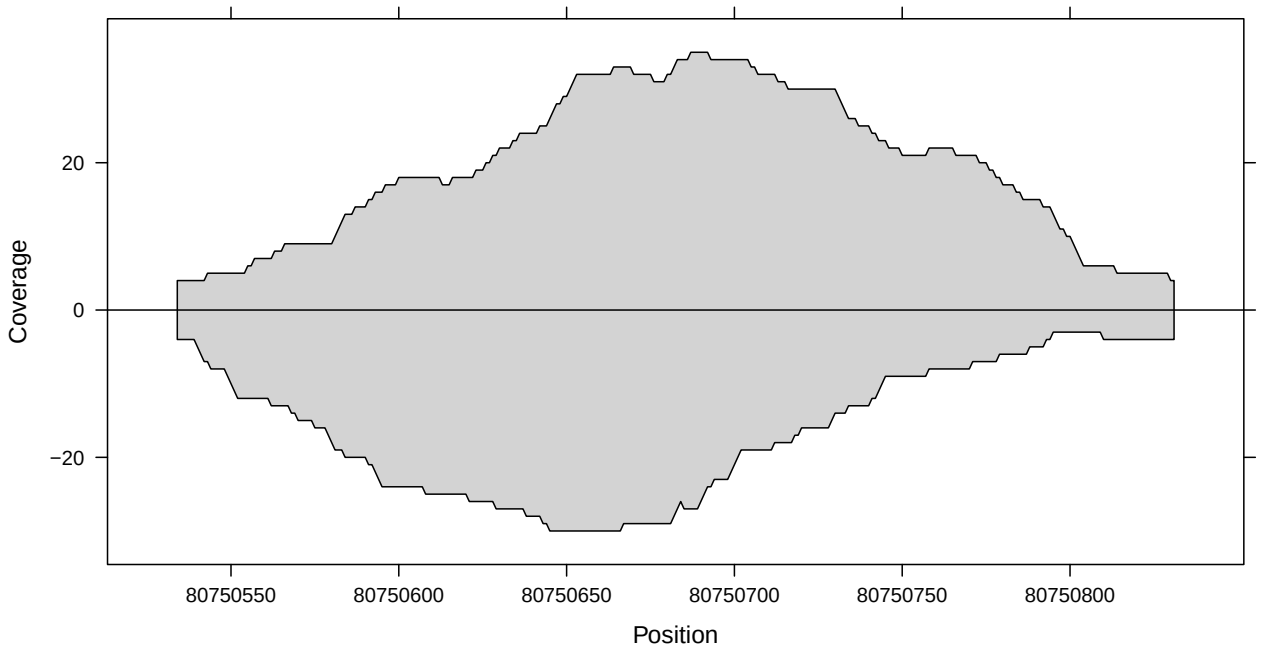

```
> coverageplot(peaks.pos[wpeaks[2]], peaks.neg[wpeaks[2]])
```



```
> coverageplot(peaks.pos[wpeaks[3]], peaks.neg[wpeaks[3]])
```




```
> coverageplot(peaks.pos[wpeaks[4]], peaks.neg[wpeaks[4]])
```

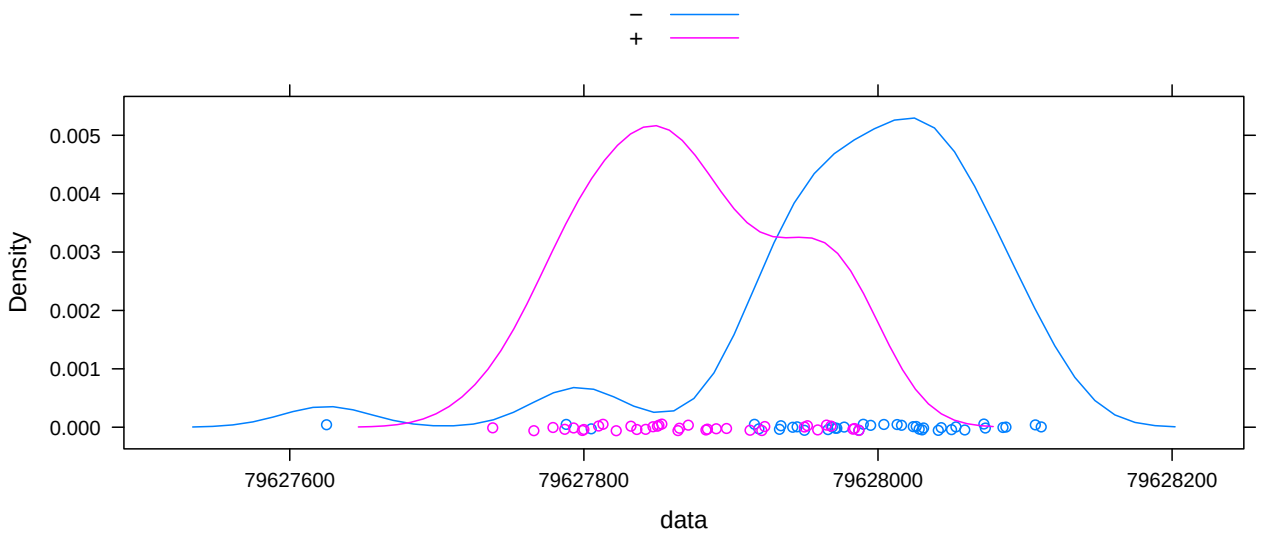


Exercise 4

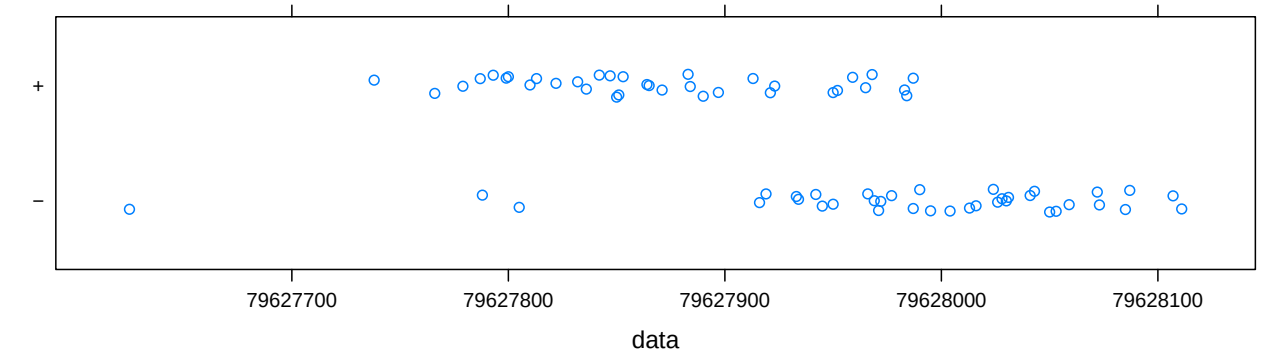
How does fragment extension length affect this picture? Change the amount of extension from 150 to 100 and 200, and reproduce these figures. What differences do you see?

It is of course also possible to plot the raw strand-level alignment locations as well.

```
> subdata <-
+     subset(do.call(make.groups, cstest$ctcf$chr10),
+            data > start(peaks)[wpeaks[1]] - 200 & data < end(peaks)[wpeaks[1]] + 200)
> densityplot(~data, data = subdata, groups = which, auto.key = TRUE)
```




```
> stripplot(which ~ data, data = subdata, jitter = TRUE)
```



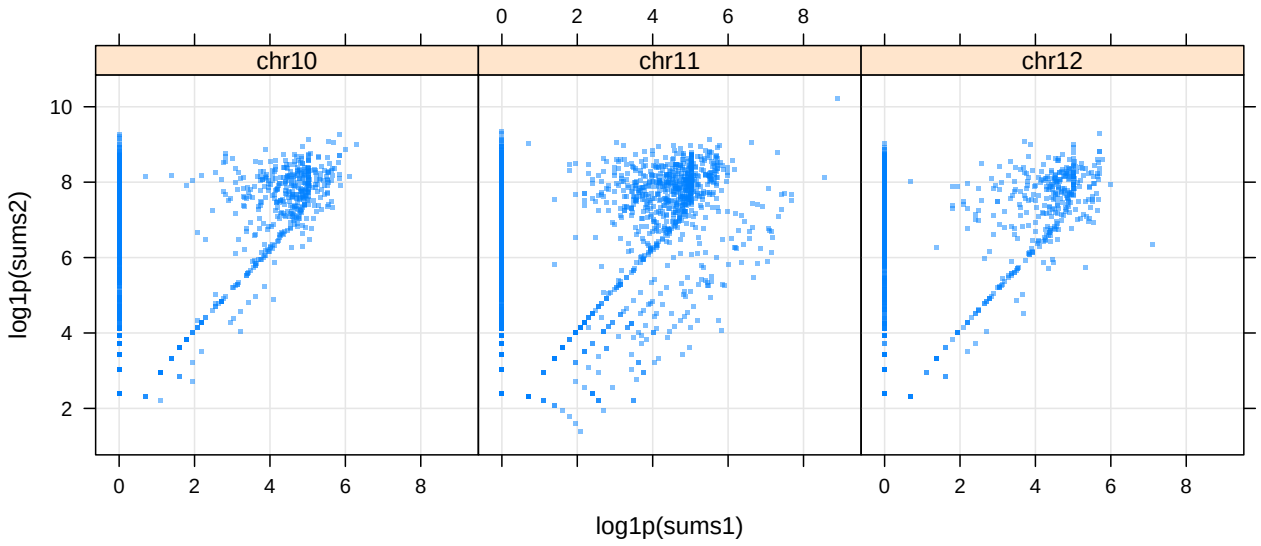
Exercise 5

Can you visually locate the likely position of TF binding sites?

Differential peaks

One common question is: which peaks are different in two samples? One simple strategy is the following: combine data from the two samples, find peaks in the combined data, and compare the contributions of the two samples.

```
> extRanges <- gdapply(cstest, extendReads, seqLen = 150)
> peakSummary <-
+     diffPeakSummary(extRanges$gfp, extRanges$ctcf,
+                     chrom.lens = mouse.chromlens, lower = 10)
> xyplot(log1p(sums2) ~ log1p(sums1) | chromosome, data = peakSummary,
+        type = c("p", "g"), alpha = 0.5, aspect = "iso", pch = ".", cex = 3)
```



In this case, the control sample has very little signal.

Version information

```
> sessionInfo()

R version 2.10.0 Under development (unstable) (2009-07-03 r48894)
x86_64-unknown-linux-gnu

locale:
[1] C

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

```
other attached packages:
[1] BSgenome.Mmusculus.UCSC.mm9_1.3.11 GenomicFeatures_0.0.5
[3] chipseq_0.1.23                     ShortRead_1.3.16
[5] lattice_0.17-25                    BSgenome_1.13.6
[7] Biostrings_2.13.21                 IRanges_1.3.26

loaded via a namespace (and not attached):
[1] Biobase_2.5.4 grid_2.10.0   hwriter_1.1   tools_2.10.0
```