Basic ChIP-Seq Data Analyis

May 31, 2009

Contents

1 Introduction

Our goal in this section of the course is to describe the use of Bioconductor software to perform some basic tasks in the analysis of ChIP-Seq data. We will use tools from the Iranges and ShortRead packages, and also use the lattice package for visualization. The next release of Bioconductor is set to include a new package called chipseq that will provide a more high-level interface to common tasks relevant for ChIP-Seq data analysis.

```
> library(ShortRead)
> library(lattice)
```

1.1 Example data

The `data` folder contains two data files, each containing data for three chromosomes from one Solexa lane, one from a CTCF mouse ChIP-Seq, and one from a GFP mouse ChIP-Seq (a background control). 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 `readAligned` function in the ShortRead package. All duplicate reads were removed and a quality score cutoff of 5 was used.

```
> load("../data/ctcf.rda")
> load("../data/gfp.rda")
```

`ctcf` and `gfp` are objects of class *AlignedRead*.

```
> ctcf

class: AlignedRead
length: 484957 reads; width: 24 cycles
chromosome: chr10 chr10 ... chr12 chr12
position: 3011944 3012936 ... 121253739 121255103
strand: - + ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

```
> gfp

class: AlignedRead
length: 316176 reads; width: 24 cycles
chromosome: chr10 chr10 ... chr12 chr12
position: 3002512 3008979 ... 121255999 121256287
strand: + - ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

Further information on each alignment can be obtained using various accessor functions whose names are hinted at in the summarized display. For example,

```
> table(strand(ctcf))

     -      +      *
240633 244324      0
```

```
> table(chromosome(gfp))

 chr10  chr11  chr12
104970 120707  90499
```

1.2 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.

1.3 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. We will extend all reads to be 150 bases long.

2 Coverage, islands, and depth

The extended aligned reads can be summarized by their *coverage*, that is, how many times each base in the genome was covered by one of these reads.

```
> cov.ctcf <- coverage(ctcf, width = mouse.chromlens, extend = 126L)
> cov.ctcf
```

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

```
> cov.ctcf$chr10
```

```
  'integer' Rle instance of length 129987169 with 310772 runs
  Lengths:  150 882 86 5 3 3 2 6 8 4 ...
  Values :  1 0 1 2 3 4 5 6 7 8 ...
```

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 can be identified by *slicing* the coverage at a depth of 1:

```
> islands <- slice(cov.ctcf$chr10, lower = 1)
> islands
```

```
  Views on a 129987169-length Rle subject

views:
            start       end width
    [1]         1       150   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [2]      1033      1403   371 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [3]      6647      6796   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [4]      8949      9098   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [5]     11202     11351   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [6]     11423     11677   255 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [7]     20769     20918   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [8]     25704     25853   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [9]     26560     26709   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    ...       ...       ...   ... ...
[99715] 126961408 126961640   233 [1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...]
[99716] 126963046 126963195   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99717] 126963758 126963907   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99718] 126966852 126967001   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99719] 126967442 126967704   263 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99720] 126968486 126968635   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99721] 126970140 126970289   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99722] 126970563 126970712   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99723] 126975203 126975352   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 area (sum) under the island, and the maximum coverage depth within that island.

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

```
[1]  150 2100  150  150  150  300
```

```
> viewMaxs(head(islands))

[1]  1 13  1  1  1  2

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

    1     2     3     4     5     6     7     8     9    10
80172 13548  2756   797   324   209   185   119   116    93

> head(depth.tab, 10)

    1     2     3     4     5     6     7     8     9    10
80230 14750  2124   472   240   184   153   121   115   107
```

Exercise 1
Repeat these steps for the `gfp` dataset.

2.1 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. We can recursively apply a summary function to all chromosomes using the `lapply` function. Here is a simple summary function that computes the frequency distribution of the number of reads per island.

```
> islandReadSummary <- function(cov)
+ {
+     s <- slice(cov, 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

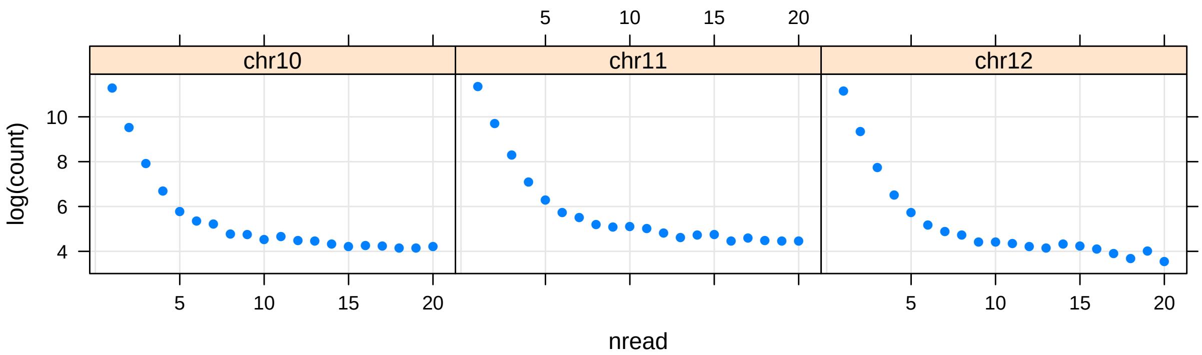
```
> head(islandReadSummary(cov.ctcf$chr10))

  nread count
1     1 80172
2     2 13548
3     3  2756
4     4   797
5     5   324
6     6   209
```

We can now use it to summarize the full dataset.

```
> nread.islands <- lapply(cov.ctcf, islandReadSummary)
> nread.islands <- do.call(make.groups, nread.islands)
> head(nread.islands)

        nread count which
chr10.1     1 80172 chr10
chr10.2     2 13548 chr10
```

```
chr10.3     3  2756 chr10
chr10.4     4   797 chr10
chr10.5     5   324 chr10
chr10.6     6   209 chr10
```

```
> xyplot(log(count) ~ nread | which, data = nread.islands,
+        subset = (nread <= 20), pch = 16, type = c("p", "g"))
```

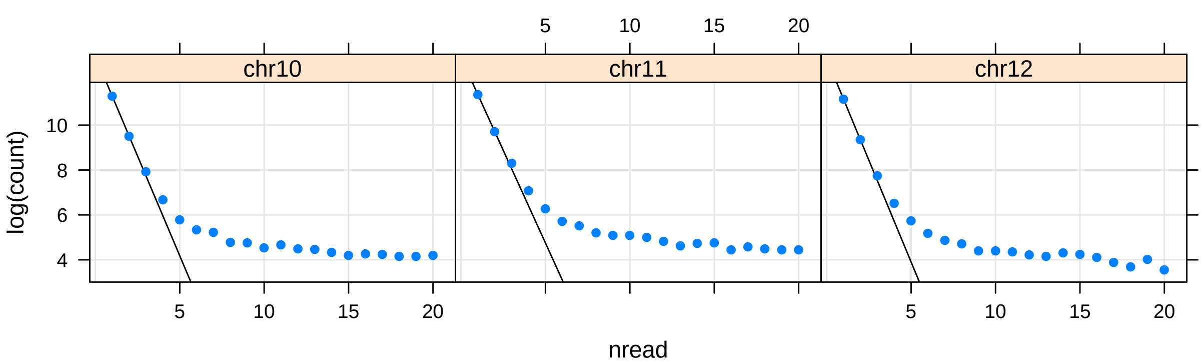


If reads were sampled randomly from the genome, then the null distribution of the number of reads per island would have a geometric distribution; that is,

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

where p is the probability a random read starts within an interval of length 150. In other words, $\log P(X = k)$ is linear in k. Although our samples are not random, we can estimate p if we assume that the islands with just one or two reads are representative of the null distribution.

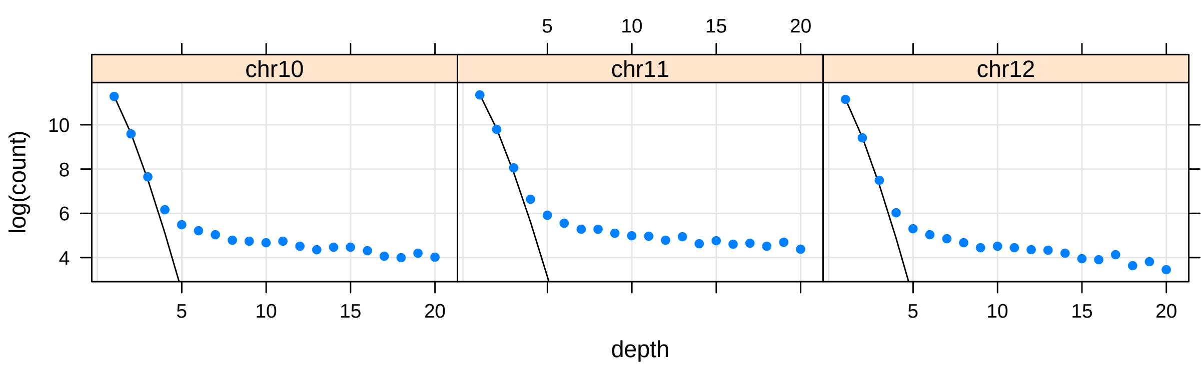
```
> xyplot(log(count) ~ nread | which, data = nread.islands,
+        subset = (nread <= 20),
+        pch = 16,
+        panel = function(x, y, ...) {
+            panel.grid(h = -1, v = -1)
+            panel.lmline(x[1:2], y[1:2], col = "black")
+            panel.xyplot(x, y, ...)
+        })
```



We can create a similar plot of the distribution of depths.

```
> islandDepthSummary <- function(cov)
+ {
+     s <- slice(cov, lower = 1)
+     tab <- table(viewMaxs(s))
+     ans <- data.frame(depth = as.numeric(names(tab)), count = as.numeric(tab))
+     ans
+ }
> depth.islands <- lapply(cov.ctcf, islandDepthSummary)
> depth.islands <- do.call(make.groups, depth.islands)
> xyplot(log(count) ~ depth | which, depth.islands,
+        subset = (depth <= 20), pch = 16,
+        panel = function(x, y, ...) {
+            panel.grid(h = -1, v = -1)
+            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, ...)
+        })
```

This assumes that the null distribution of depths has a Poisson distribution, which is not strictly true, but seems to give a reasonble fit.



Exercise 2
Produce similar plots for the `gfp` *dataset. What qualitatitve differences do you see? Based on your findings, 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?*

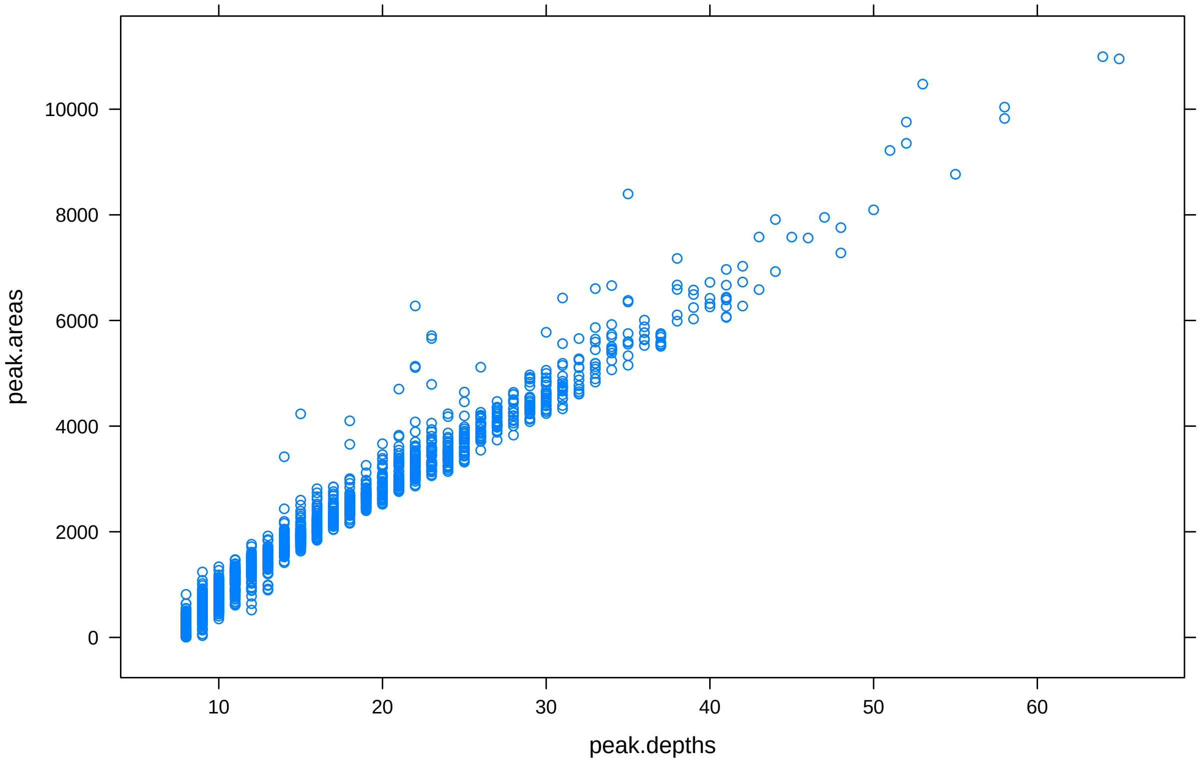
2.2 Peaks

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

```
> peaks <- slice(cov.ctcf$chr10, lower = 8)
> peaks

  Views on a 129987169-length Rle subject

views:
           start       end width
   [1]      1146      1287   142 [ 8  8  8  8  9 10 11 11 11 11 11 11 11 ...]
   [2]    222982    223074    93 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [3]    258257    258261     5 [8 8 8 8 8]
   [4]    258266    258443   178 [ 8  8  8  8  8  9  9  9  9  9  9 10 11 ...]
   [5]    265866    265999   134 [ 8  8  8  8  8  8  8  8  8  8  8  8  9 ...]
   [6]    449049    449111    63 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [7]    606027    606130   104 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
   [8]    639945    640155   211 [ 8  8 10 10 10 12 12 12 12 12 12 12 12 ...]
   [9]   1298612   1298858   247 [ 8  9  9 10 10 10 11 11 11 11 11 11 12 ...]
   ...       ...       ...   ... ...
[1746] 125974702 125974806   105 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 ...]
[1747] 125974827 125974830     4 [8 8 8 8]
[1748] 125974835 125974835     1 [8]
[1749] 126047124 126047135    12 [8 8 8 8 8 8 8 8 8 8 8 8]
[1750] 126518227 126518373   147 [ 8  8  8  8  8  8  8  8  9  9  9  9  9 ...]
[1751] 126521514 126521564    51 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1752] 126653571 126653753   183 [ 8  8  8  8  8  8  8  9 10 10 11 11 11 ...]
[1753] 126654948 126655088   141 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 ...]
[1754] 126738854 126738991   138 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
```

Interesting properties of peaks are their maximum depth and area under the peak (a relative measure of how localized the peak is).

```
> peak.depths <- viewMaxs(peaks)
> peak.areas <- viewSums(peaks)
> xyplot(peak.areas ~  peak.depths)
```



Exercise 3

Produce a similar plot for the `gfp` *dataset. What differences do you see, particularly in terms of the number of peaks and the distribution of depths?*

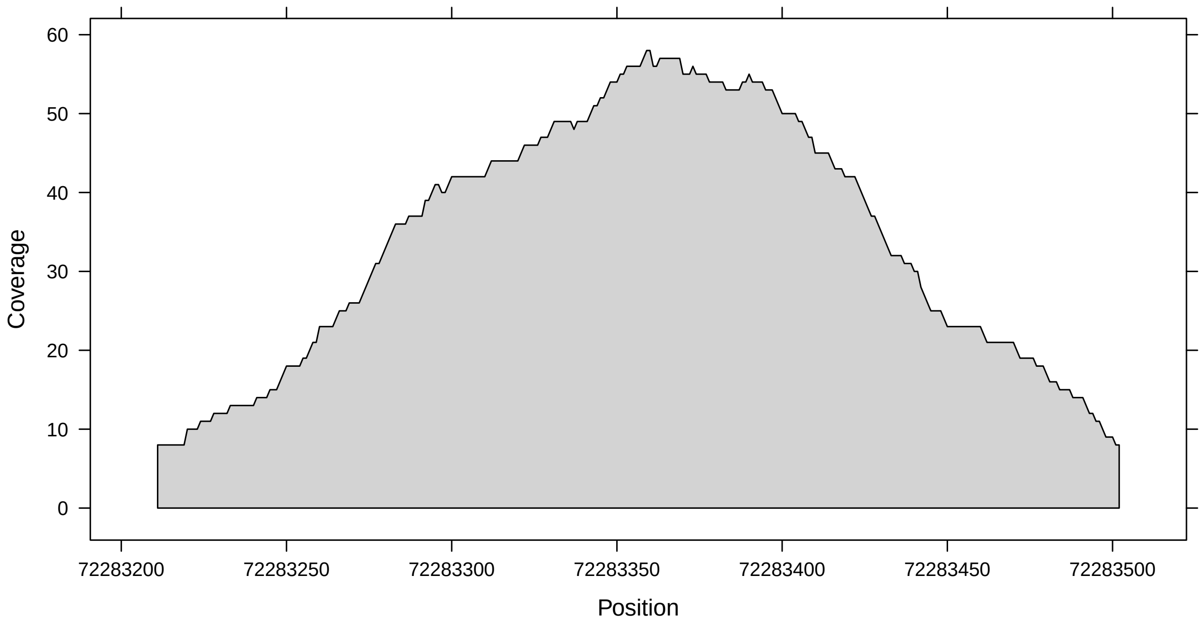
We can order the peaks by depth

```
> wpeaks <- tail(order(peak.depths), 4)
> peaks[wpeaks]

  Views on a 129987169-length Rle subject

views:
        start       end width
[1]  72283211  72283502   292 [ 8  8  8  8  8  8  8  8  8 10 10 10 10 11 ...]
[2] 123344361 123344655   295 [ 8  8  8  8  8  8  8 10 10 10 10 11 11 11 ...]
[3]  74863897  74864200   304 [ 8  8  9  9  9  9 10 10 11 10 10 10 10 10 ...]
[4]  77738717  77739014   298 [ 8  8  8  8  8  8  9 10 11 12 13 13 13 13 ...]
```

and plot individual peaks using this function:

```
> coverageplot <- function (peaks, xlab = "Position", ylab = "Coverage", ...)
+ {
+     pos1 <- seq(start(peaks[1]), end(peaks[1]))
+     cov1 <- as.integer(peaks[[1]])
+     pos1 <- c(head(pos1, 1), pos1, tail(pos1, 1))
+     cov1 <- c(0, cov1, 0)
+     xyplot(cov1 ~ pos1, ..., panel = panel.polygon,
```

```
+             col = "lightgrey", xlab = xlab, ylab = ylab)
+
+ }

> coverageplot(peaks[wpeaks[1]])
```



Exercise 4
How does the amount by which each read is extended affect the analysis? In calls to `coverage`*, We have used* `extend=126L` *to get a total length of 150 for each read. Try lengths of 100 and 200 and see how the results change.*

3 Version information

- R version 2.9.0 Patched (2009-05-27 r48659), `x86_64-unknown-linux-gnu`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: BSgenome 1.12.0, BSgenome.Mmusculus.UCSC.mm9 1.3.11, Biostrings 2.12.3, IRanges 1.2.2, ShortRead 1.2.0, lattice 0.17-25
- Loaded via a namespace (and not attached): Biobase 2.4.1, grid 2.9.0, hwriter 1.1