

Some Basic Analysis of ChIP-Seq Data

January 23, 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 as-yet-unreleased `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.Mmusculus.UCSC.mm9)
> library(lattice)
```

Example data

The `cstest` data set is included in the `chipseq` package to help demonstrate its capabilities. 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`. 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
```

```
GenomeDataList of length 2
names(2): ctcf gfp
```

`cstest` is an object of class “*GenomeDataList*”, and has a list-like structure, each component 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 3045044
$ +: int [1:73179] 3012936 3012941 3012944 3012955 3012963 3012969 3012978 3013071 3018464 3020764
```

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.

We extend all reads to be 200 bases long. This is done using the `extendReads()` function, which can work on data from one chromosome in one lane.

```
> ext <- extendReads(cstest$ctcf$chr10, seqLen = 200)
> head(ext)
```

```
IRanges of length 6
      start      end width
[1] 3012936 3013135   200
[2] 3012941 3013140   200
[3] 3012944 3013143   200
[4] 3012955 3013154   200
[5] 3012963 3013162   200
[6] 3012969 3013168   200
```

The result is essentially a collection of intervals (ranges) over the reference genome.

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 of length 129993255 with 288928 runs
 Lengths:  3012799 97 2 37 5 3 11 8 6 9 ...
 Values  :  0 1 2 3 5 6 7 8 9 10 ...
```

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] 3012800 3013270 471 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[2] 3018464 3018663 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[3] 3020766 3020965 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[4] 3023019 3023218 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[5] 3023240 3023439 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[6] 3032536 3032735 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[7] 3038377 3038576 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[8] 3040312 3040554 243 [1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 ...]
[9] 3041098 3041297 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
...
[87957] 129973175 129973447 273 [1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...]
[87958] 129974813 129975012 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87959] 129975575 129975774 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87960] 129978669 129978868 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87961] 129979209 129979571 363 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87962] 129980253 129980452 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87963] 129981957 129982156 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87964] 129982330 129982529 200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[87965] 129987020 129987219 200 [1 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] 2400 200 200 200 200 200
```

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

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

```
> nread.tab <- table(viewSums(islands) / 200)
```

```
> depth.tab <- table(viewMaxs(islands))
```

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

```
      1      2      3      4      5      6      7      8      9     10
68111 13350 3022  925  415  247  191  122  132  100
```

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

```
      1      2      3      4      5      6      7      8      9     10
68159 14745 2388  547  256  180  150  129  120  102
```

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 = 200)
+   s <- slice(coverage(g), lower = 1)
+   tab <- table(viewSums(s) / 200)
+   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))
```

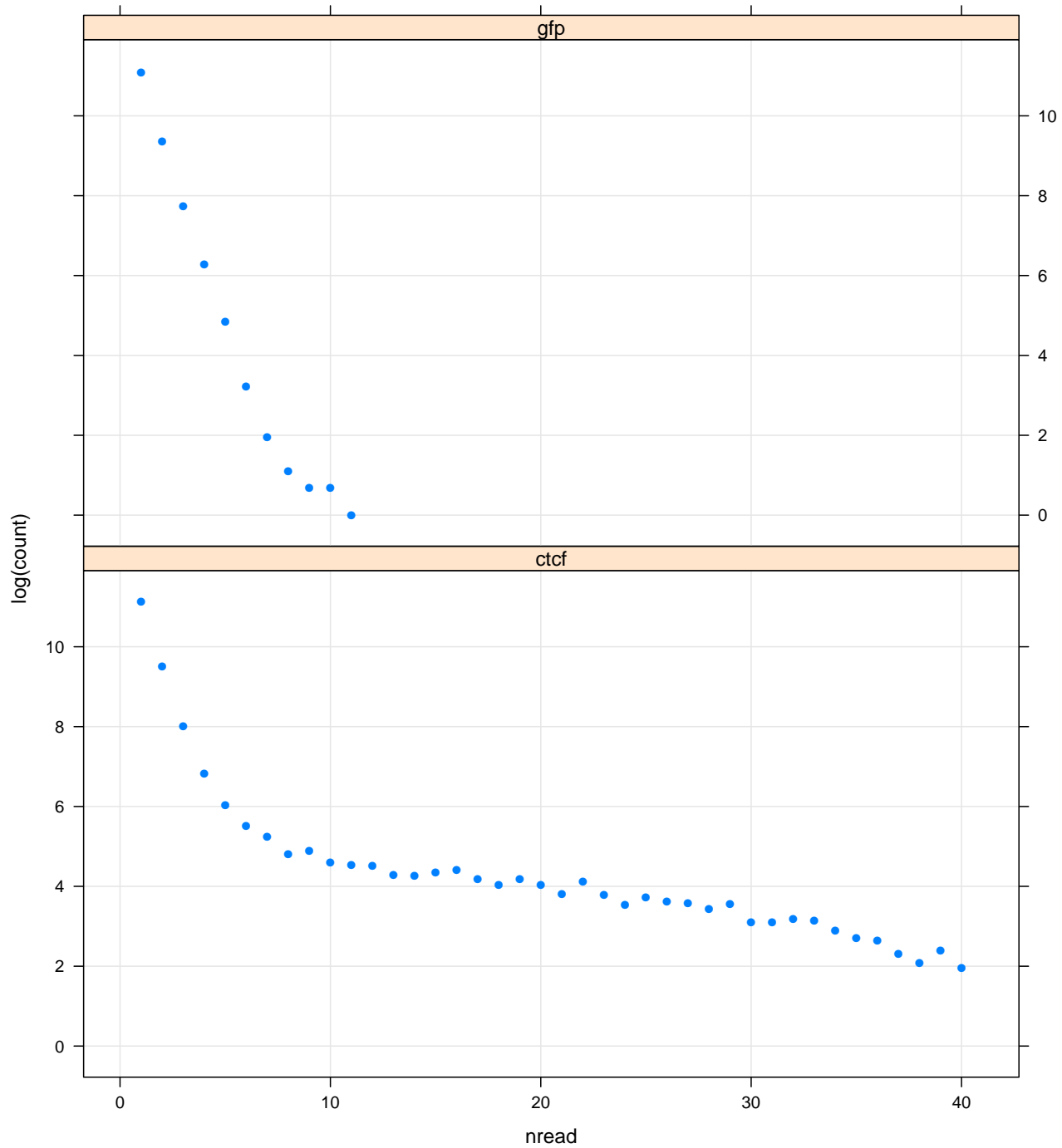
	nread	count
1	1	68111
2	2	13350
3	3	3022
4	4	925
5	5	415
6	6	247

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	68111	chr10	ctcf
2	2	13350	chr10	ctcf
3	3	3022	chr10	ctcf
4	4	925	chr10	ctcf
5	5	415	chr10	ctcf
6	6	247	chr10	ctcf

```
> xyplot(log(count) ~ nread | sample, nread.islands,  
+       subset = (chromosome == "chr10" & nread <= 40),  
+       layout = c(1, 2), pch = 16, type = c("p", "g"))
```

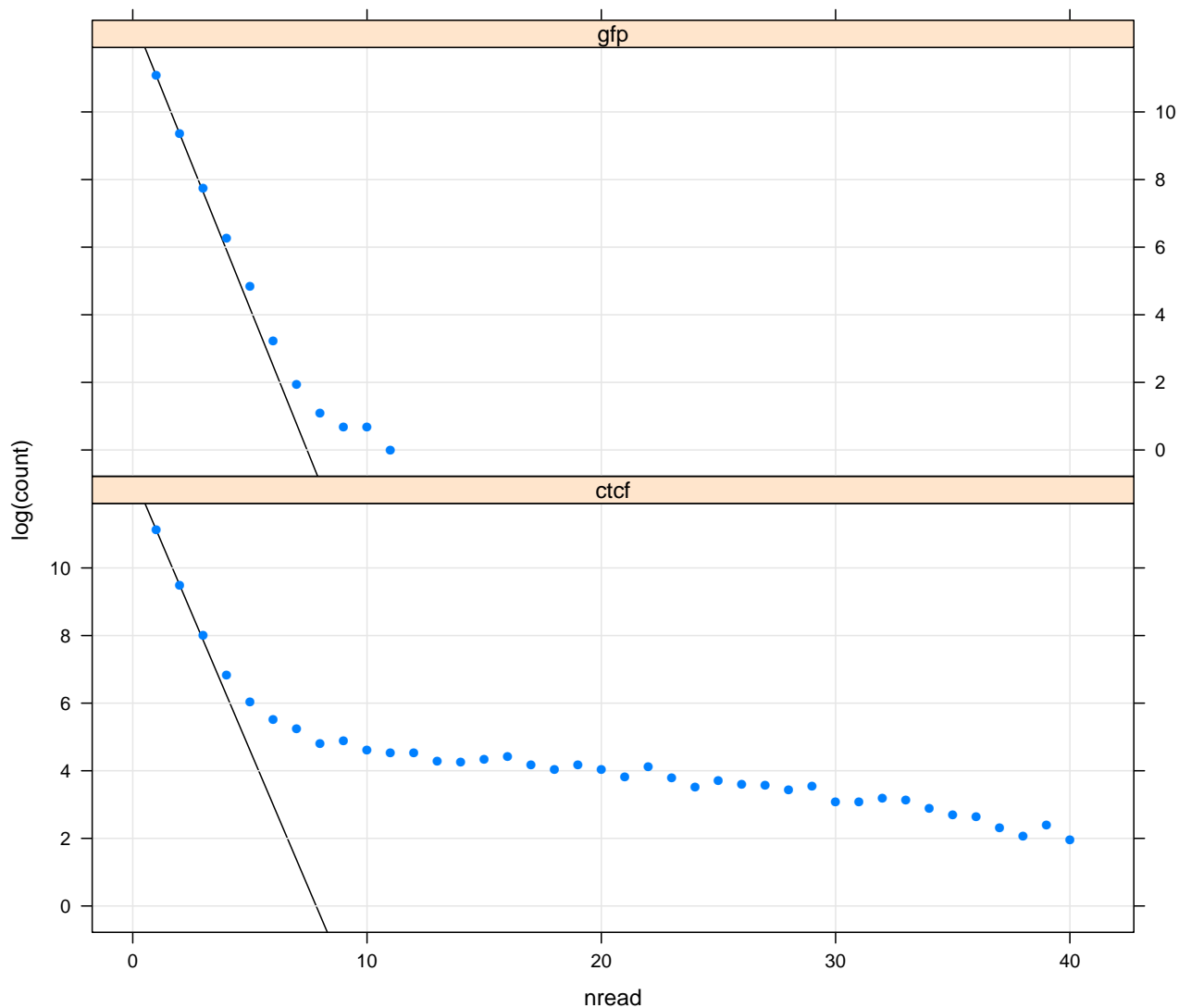


If reads were sampled randomly from the genome, then the null distribution number of reads per island would have a geometric distribution; that is,

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

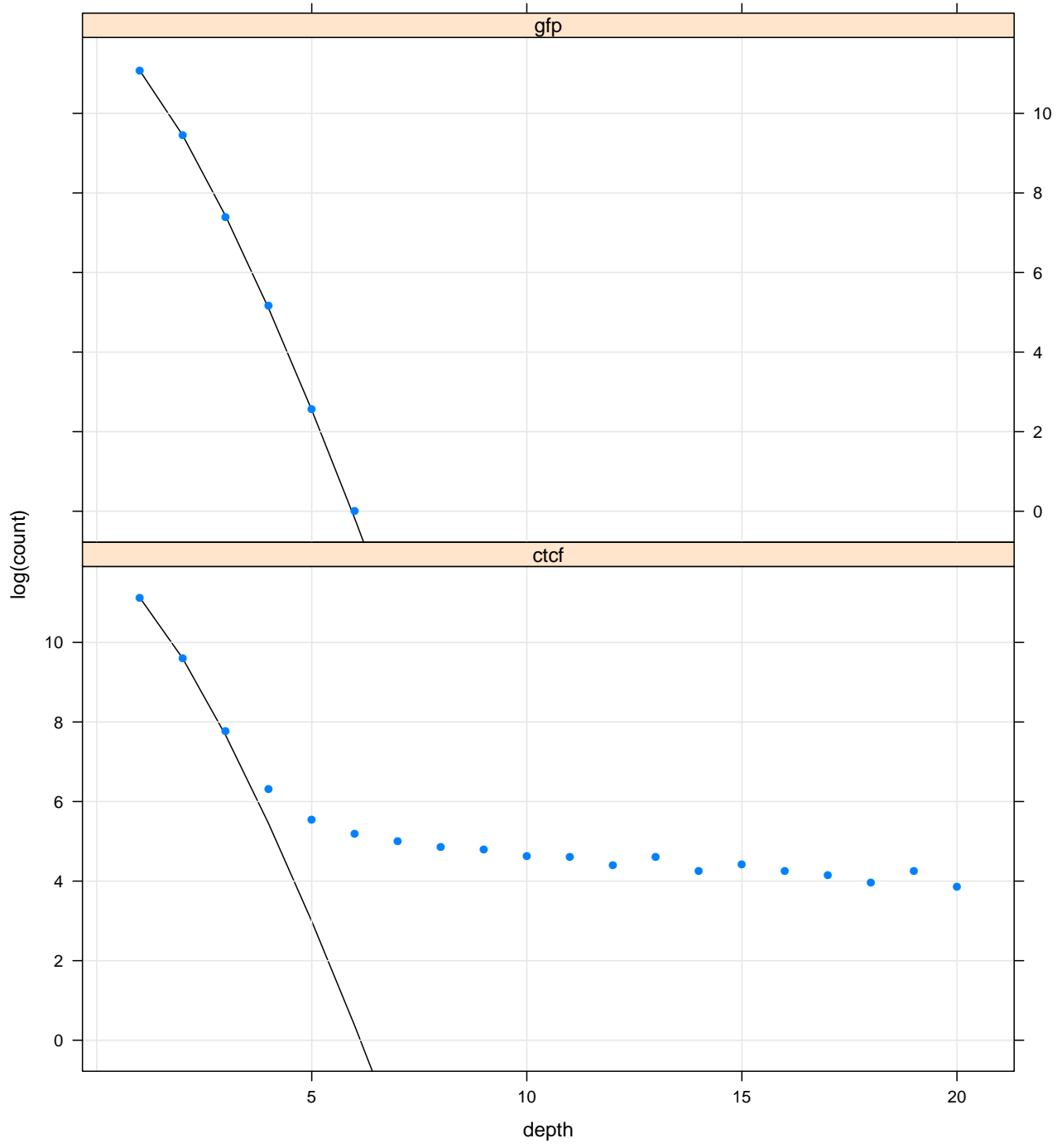
In other words, $\log P(X = k)$ is linear in k . Although our samples are not random, the islands with just one or two reads may be representative of the null distribution.

```
> xyplot(log(count) ~ nread | sample, nread.islands,
+       subset = (chromosome == "chr10" & nread <= 40),
+       layout = c(1, 2), 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.

```
> islandDepthSummary <- function(x)
+ {
+   g <- extendReads(x, seqLen = 200)
+   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 | sample, depth.islands,
+   subset = (chromosome == "chr10" & depth <= 20),
+   layout = c(1, 2), 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, ...)
+   })
```



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] 3012955 3013135 181 [ 8 8 8 8 8 8 8 8 8 9 9 9 9 9 ...]
[2] 3234799 3234895  97 [ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[3] 3270012 3270297 286 [8 8 8 9 9 9 9 9 8 8 8 8 8 8 8 8 8 8 ...]
[4] 3277662 3277832 171 [ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[5] 3277848 3277859  12 [8 8 8 8 8 8 8 8 8 8 8 8]
[6] 3460857 3460973 117 [ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[7] 3617850 3617983 134 [ 8 8 8 8 8 8 8 8 9 9 9 9 10 10 ...]
[8] 3651712 3651992 281 [ 8 8 9 9 9 10 10 10 10 10 10 10 10 ...]
[9] 4310402 4310712 311 [ 8 8 8 9 9 9 9 9 9 9 9 9 10 10 ...]
...
[1747] 128986519 128986595  77 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1748] 128986604 128986610   7 [8 8 8 8 8 8]
[1749] 128986638 128986673  36 [8 8 8 8 8 8 9 9 9 9 8 8 8 8 9 9 9 8 8 8 ...]
[1750] 129058889 129058980  92 [8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 ...]
[1751] 129530031 129530201 171 [ 8 8 9 9 9 9 9 9 9 9 9 9 9 9 ...]
[1752] 129533303 129533381  79 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1753] 129665351 129665586 236 [ 8 9 9 9 9 9 9 9 9 9 10 10 10 10 ...]
[1754] 129666784 129666947 164 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1755] 129750671 129750849 179 [ 8 8 8 8 8 8 9 9 9 9 9 9 9 9 ...]
```

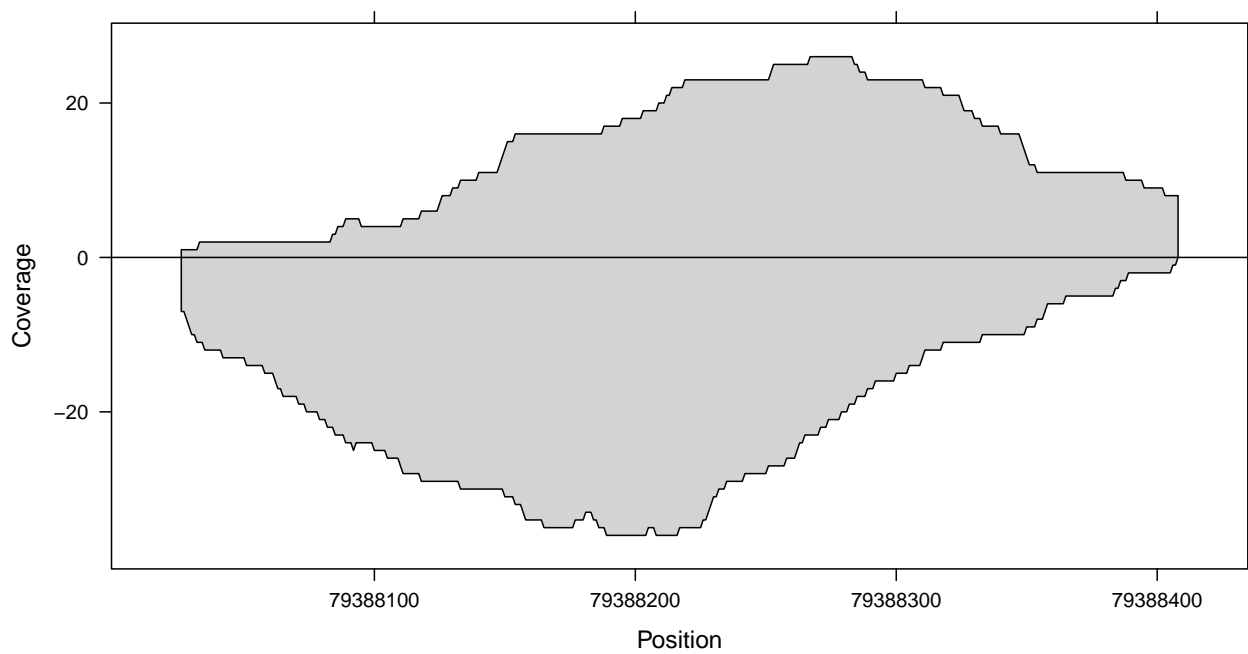
It is meaningful to ask about the contribution of each strand to each peak, as the sequenced region of pull-down fragments would be on opposite sides of a binding site depending on which strand it matched. We can compute strand-specific coverage, and look at the individual coverages under the combined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

```
> peak.depths <- viewMaxs(peaks)
> cov.pos <- coverage(extendReads(cstest$ctcf$chr10, strand = "+", seqLen = 200),
+                    width = mouse.chromlens["chr10"])
> cov.neg <- coverage(extendReads(cstest$ctcf$chr10, strand = "-", seqLen = 200),
+                    width = mouse.chromlens["chr10"])
> peaks.pos <- copyIRanges(peaks, cov.pos)
> peaks.neg <- copyIRanges(peaks, cov.neg)
> wpeaks <- tail(order(peak.depths), 4)
> wpeaks
```

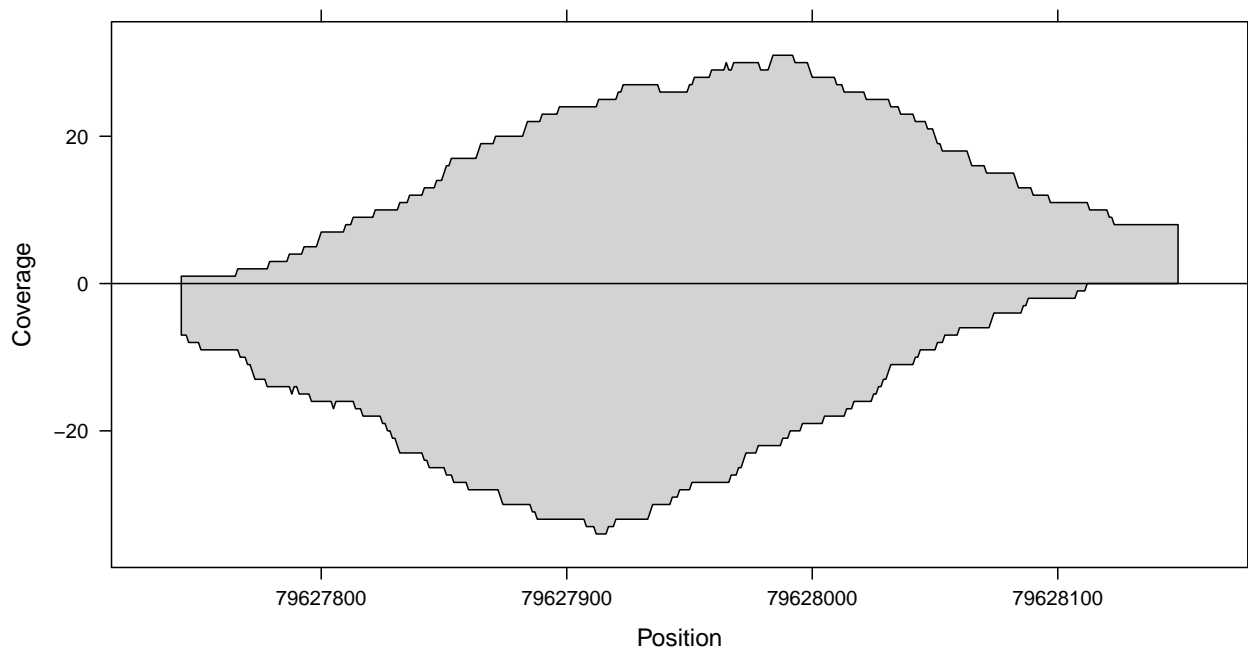
```
[1] 971 989 1079 922
```

We plot four highest peaks below.

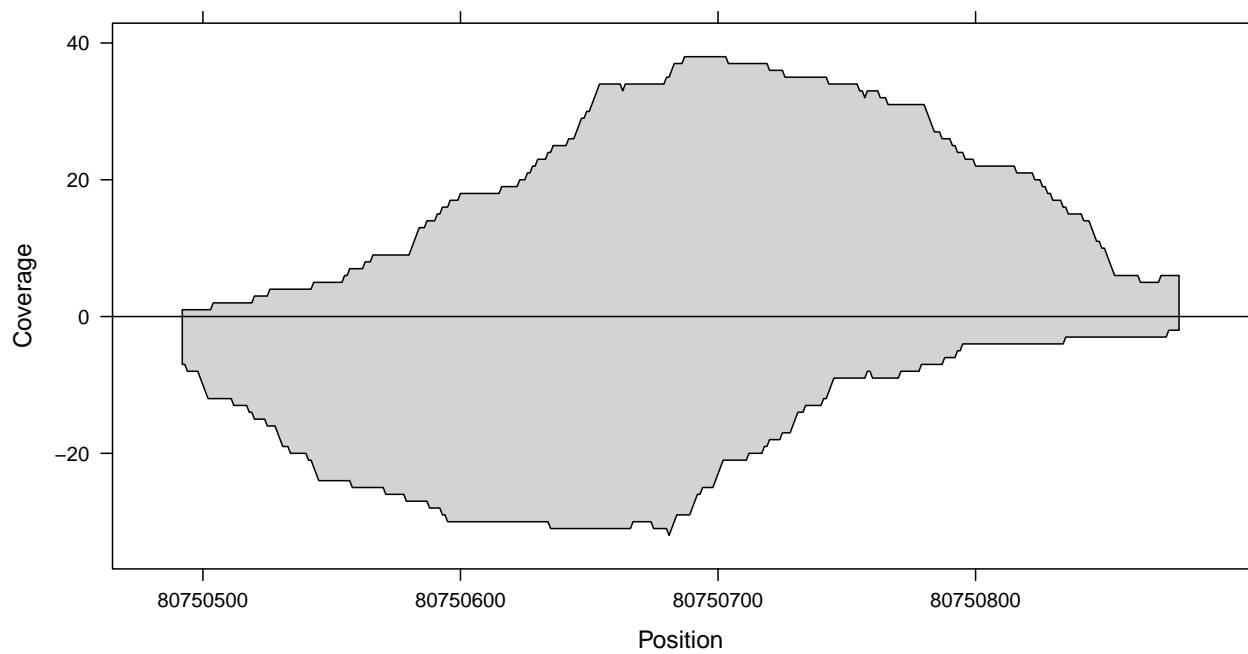
```
> coverageplot(peaks.pos[wpeaks[1]], peaks.neg[wpeaks[1]])
```



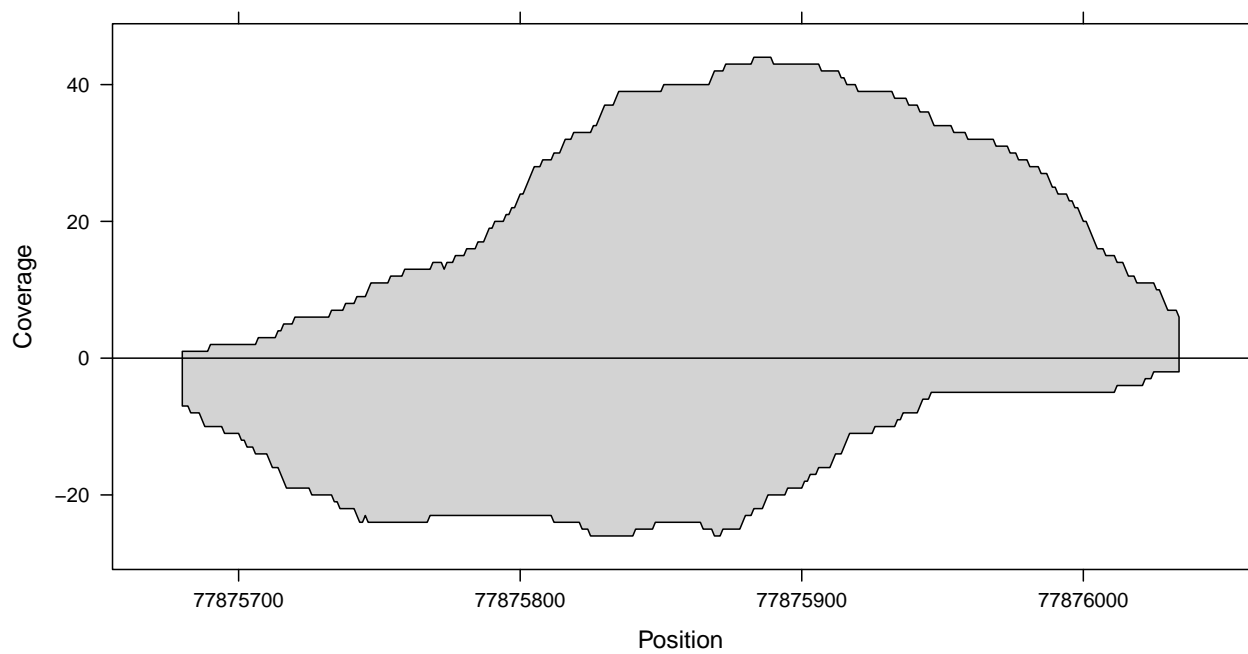
```
> coverageplot(peaks.pos[wpeaks[2]], peaks.neg[wpeaks[2]])
```



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



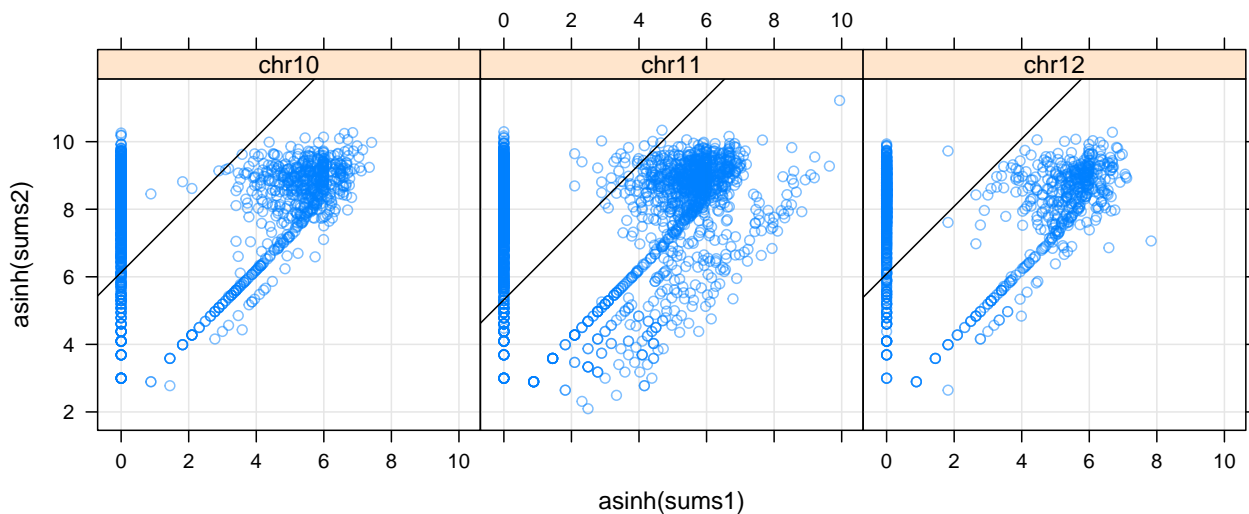
```
> coverageplot(peaks.pos[wpeaks[4]], peaks.neg[wpeaks[4]])
```



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 = 200)
> peakSummary <-
+   diffPeakSummary(extRanges$gfp, extRanges$ctcf,
+                   chrom.lens = mouse.chromlens, lower = 10)
> xyplot(asinh(sums2) ~ asinh(sums1) | chromosome,
+        data = peakSummary,
+        ## subset = (chromosome %in% c("chr1", "chr2")),
+        panel = function(x, y, ...) {
+          panel.xyplot(x, y, ...)
+          panel.abline(median(y - x), 1)
+        },
+        type = c("p", "g"), alpha = 0.5, aspect = "iso")
```



We use a simple cutoff to flag peaks that are different.

```
> peakSummary <-
+   within(peakSummary,
+         {
+           diffs <- asinh(sums2) - asinh(sums1)
+           resids <- (diffs - median(diffs)) / mad(diffs)
+           up <- resids > 2
+           down <- resids < -2
+         })
> head(peakSummary)
```

	chromosome	start	end	comb.max	overlap1	overlap2	sums1	sums2	maxs1	maxs2
1	chr10	3012969	3013098	11	0	12	0	1348	0	11
2	chr10	3234832	3234882	10	0	10	0	510	0	10
3	chr10	3270059	3270295	20	1	23	158	3614	1	19
4	chr10	3277695	3277775	13	0	13	0	924	0	13
5	chr10	3460894	3460924	10	0	10	0	310	0	10
6	chr10	3617861	3617909	10	0	10	0	490	0	10

	down	up	resids	diffs
1	FALSE	FALSE	0.5247940	7.899525
2	FALSE	FALSE	0.2834625	6.927559
3	FALSE	FALSE	-0.6594503	3.129965
4	FALSE	FALSE	0.4310227	7.521860
5	FALSE	FALSE	0.1598535	6.429722
6	FALSE	FALSE	0.2735295	6.887554

Placing peaks in genomic context

Locations of individual peaks may be of interest. Alternatively, a global summary might look at classifying the peaks of interest in the context of genomic features such as promoters, upstream regions, etc. The `geneMouse` function defined in the `GenomicFeatures.Mmusculus.UCSC.mm9` package returns gene location information from UCSC. The `genomic_regions` function converts this into a set of ranges defining promoters, upstream regions, etc.

```
> gregions <- transcripts(genes = geneMouse(), proximal = 2000)
> ## gregions$gene <- as.character(gregions$gene)
> gregions
```

RangedData with 43856 rows and 5 value columns across 33 spaces

	space	ranges	gene	promoter
	<character>	<IRanges>	<character>	<IRanges>
1	chr1	[3195985, 3205713]	uc007aet.1	[3203714, 3207713]
2	chr1	[3204563, 3661579]	uc007aeu.1	[3659580, 3663579]
3	chr1	[3638392, 3648985]	uc007aev.1	[3646986, 3650985]
4	chr1	[4280927, 4399322]	uc007aew.1	[4397323, 4401322]
5	chr1	[4334224, 4350473]	uc007aex.1	[4348474, 4352473]
6	chr1	[4481009, 4483816]	uc007aey.1	[4481817, 4485816]
7	chr1	[4481009, 4486494]	uc007aez.1	[4484495, 4488494]
8	chr1	[4764015, 4775768]	uc007afd.1	[4773769, 4777768]
9	chr1	[4766458, 4775768]	uc007aff.1	[4773769, 4777768]
10	chr1	[4797974, 4832908]	uc007afg.1	[4795974, 4799973]

	threeprime	upstream	downstream
	<IRanges>	<IRanges>	<IRanges>
1	[3193985, 3197984]	[3207714, 3215713]	[3185985, 3193984]
2	[3202563, 3206562]	[3663580, 3671579]	[3194563, 3202562]
3	[3636392, 3640391]	[3650986, 3658985]	[3628392, 3636391]
4	[4278927, 4282926]	[4401323, 4409322]	[4270927, 4278926]
5	[4332224, 4336223]	[4352474, 4360473]	[4324224, 4332223]
6	[4479009, 4483008]	[4485817, 4493816]	[4471009, 4479008]

```

7 [4479009, 4483008] [4488495, 4496494] [4471009, 4479008]
8 [4762015, 4766014] [4777769, 4785768] [4754015, 4762014]
9 [4764458, 4768457] [4777769, 4785768] [4756458, 4764457]
10 [4830909, 4834908] [4787974, 4795973] [4834909, 4842908]
...
<43846 more rows>

```

This can be used to obtain a tabulation of the peaks.

```

> ans <- contextDistribution(peakSummary, gregions)
> head(ans)

```

```

  type total gene promoter threeprime upstream downstream chromosome
1 all  1544  746    233    182    294    313    chr10
2 up     0    0     0     0     0     0     chr10
3 down   0    0     0     0     0     0     chr10
4 all  2499 1367    428    387    629    740    chr11
5 up     0    0     0     0     0     0     chr11
6 down   0    0     0     0     0     0     chr11

```

```

> sumtab <-
+   as.data.frame(rbind(total = xtabs(total ~ type, ans),
+                         promoter = xtabs(promoter ~ type, ans),
+                         threeprime = xtabs(threeprime ~ type, ans),
+                         upstream = xtabs(upstream ~ type, ans),
+                         downstream = xtabs(downstream ~ type, ans),
+                         gene = xtabs(gene ~ type, ans)))
> cbind(sumtab, ratio = round(sumtab[, "down"] / sumtab[, "up"], 3))

```

```

      all up down ratio
total  5180 0  0  NaN
promoter  801 0  0  NaN
threeprime 655 0  0  NaN
upstream 1080 0  0  NaN
downstream 1242 0  0  NaN
gene  2652 0  0  NaN

```

Version information

```

> sessionInfo()

```

```

R version 2.10.0 (2009-10-26)
x86_64-unknown-linux-gnu

```

```

locale:

```

```

[1] LC_CTYPE=en_US.UTF-8    LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8     LC_COLLATE=en_US.UTF-8
[5] LC_MONETARY=C           LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8    LC_NAME=C

```

```
[9] LC_ADDRESS=C          LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

attached base packages:

```
[1] tools      stats      graphics  grDevices  utils      datasets  methods
[8] base
```

other attached packages:

```
[1] BSgenome.Mmusculus.UCSC.mm9_1.3.16
[2] GenomicFeatures.Mmusculus.UCSC.mm9_0.1.0
[3] GenomicFeatures_0.2.0
[4] rtracklayer_1.6.0
[5] RCurl_1.3-0
[6] bitops_1.0-4.1
[7] chipseq_0.2.1
[8] ShortRead_1.4.0
[9] lattice_0.17-26
[10] BSgenome_1.14.2
[11] Biostrings_2.14.7
[12] IRanges_1.4.7
```

loaded via a namespace (and not attached):

```
[1] Biobase_2.6.0 DBI_0.2-4      grid_2.10.0   hwriter_1.1   RSQLite_0.7-3
[6] XML_2.6-0
```