Some Basic Analyis 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(lattice)
```

Example data

The `alignedLocs` data set, provided in the file `alignedLocs.rda`, contains data from three Solexa lanes. 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).

```
> load("../data/alignedLocs.rda")
> alignedLocs

  A GenomeDataList instance of length 3
```

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

```
> alignedLocs$sample1

  A GenomeData instance of length 21
```

Each of these are themselves lists:

```
> str(head(as(alignedLocs$sample1, "list")))

List of 6
 $ chr1 :List of 2
  ..$ -: int [1:50584] 3013802 3026308 3035525 3039158 3060684 3074079 3094661 3103669 3104501 3107344 ...
  ..$ +: int [1:50181] 3001001 3018535 3041392 3055167 3064081 3067760 3084702 3087437 3088347 3102491 ...
 $ chr10:List of 2
  ..$ -: int [1:38604] 3013702 3019386 3021977 3022708 3031335 3032387 3033386 3033860 3035734 3037785 ...
  ..$ +: int [1:38627] 3012735 3013613 3019737 3020950 3022283 3022648 3024694 3027070 3032179 3032304 ...
 $ chr11:List of 2
  ..$ -: int [1:40791] 3000457 3001529 3001773 3001844 3001844 3001936 3003649 3004101 3004253 3004647 ...
  ..$ +: int [1:40733] 3000450 3000454 3000478 3001160 3001863 3002157 3003198 3003662 3003834 3004667 ...
 $ chr12:List of 2
  ..$ -: int [1:31357] 3009452 3028115 3050883 3098914 3109935 3109974 3109975 3110013 3110014 3110015 ...
  ..$ +: int [1:31469] 3001799 3017578 3082192 3084558 3084558 3109902 3109963 3110012 3110012 3110015 ...
 $ chr13:List of 2
```

```
  ..$ -: int [1:27449] 3006590 3027314 3035978 3036441 3044729 3044729 3045762 3055107 3061799 3066553 ...
  ..$ +: int [1:27330] 3005906 3005906 3007903 3064610 3087709 3118497 3122771 3127866 3138624 3154032 ...
 $ chr14:List of 2
  ..$ -: int [1:27447] 4008628 4075161 4283048 4368538 4387943 4402581 4444197 4507079 4696912 4925156 ...
  ..$ +: int [1:27469] 4148048 4214070 4256133 4361784 4387792 5970011 6012807 6036360 6036360 6046029 ...
```

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 (35 in our data) 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(alignedLocs$sample1$chr5, seqLen = 200)
> head(ext)
```

```
IRanges object:
    start     end width
1 3018808 3019007   200
2 3025021 3025220   200
3 3030072 3030271   200
4 3041363 3041562   200
5 3047643 3047842   200
6 3052628 3052827   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, start = 1, end = mouse.chromlens["chr5"])
> cov
```

```
  'integer' Rle instance of length 152537259 with 182534 runs
  Lengths:  3018807 200 5852 161 39 161 4851 200 6693 200 ...
  Values :  0 1 0 1 2 1 0 1 0 1 ...
```

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 152537259-length Rle subject

views:
            start       end width
    [1]   3018808   3019007   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [2]   3024860   3025220   361 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [3]   3030072   3030271   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [4]   3036965   3037164   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [5]   3041363   3041562   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [6]   3047643   3047842   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [7]   3052628   3052827   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [8]   3071085   3071284   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    [9]   3075285   3075484   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    ...       ...       ...   ... ...
[67994] 152516119 152516318   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[67995] 152517628 152517827   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[67996] 152521031 152521230   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[67997] 152521997 152522196   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[67998] 152527104 152527410   307 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[67999] 152528326 152528525   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[68000] 152528898 152529097   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[68001] 152530611 152530810   200 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[68002] 152532557 152532756   200 [1 1 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] 200 400 200 200 200 200

> viewMaxs(head(islands))

[1] 1 2 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
52396 10885  2855   901   355   174    94    71    50    33

> head(depth.tab, 10)

    1     2     3     4     5     6     7     8     9    10
52423 11647  2652   626   234   101    76    47    32    25
```

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, 1, max(end(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(alignedLocs$sample1$chr5))
```

```
  nread count
1     1 52396
2     2 10885
3     3  2855
4     4   901
5     5   355
6     6   174
```

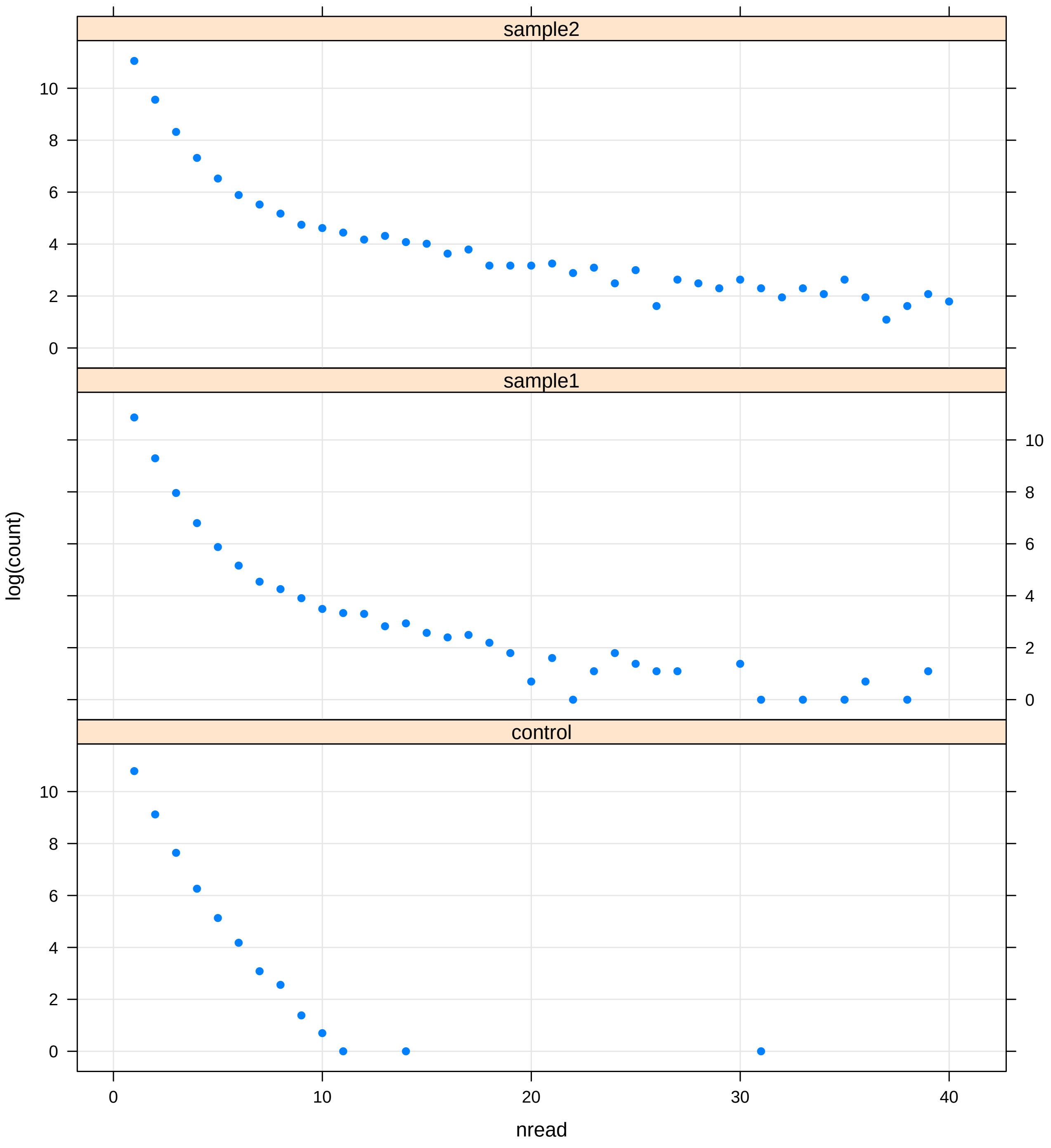
We can now use it to summarize the full dataset.

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

```
  nread count chromosome  sample
1     1 61711       chr1 control
2     2 10117       chr1 control
3     3  2033       chr1 control
4     4   487       chr1 control
5     5   144       chr1 control
6     6    46       chr1 control
```

A one-step interface is also available, but may go away in future.

```
> nread.islands <- summarizeReads(alignedLocs, summary.fun = islandReadSummary)
> xyplot(log(count) ~ nread | sample, nread.islands,
+        subset = (chromosome == "chr5" & nread <= 40),
+        layout = c(1, 3), pch = 16, type = c("p", "g"))
```

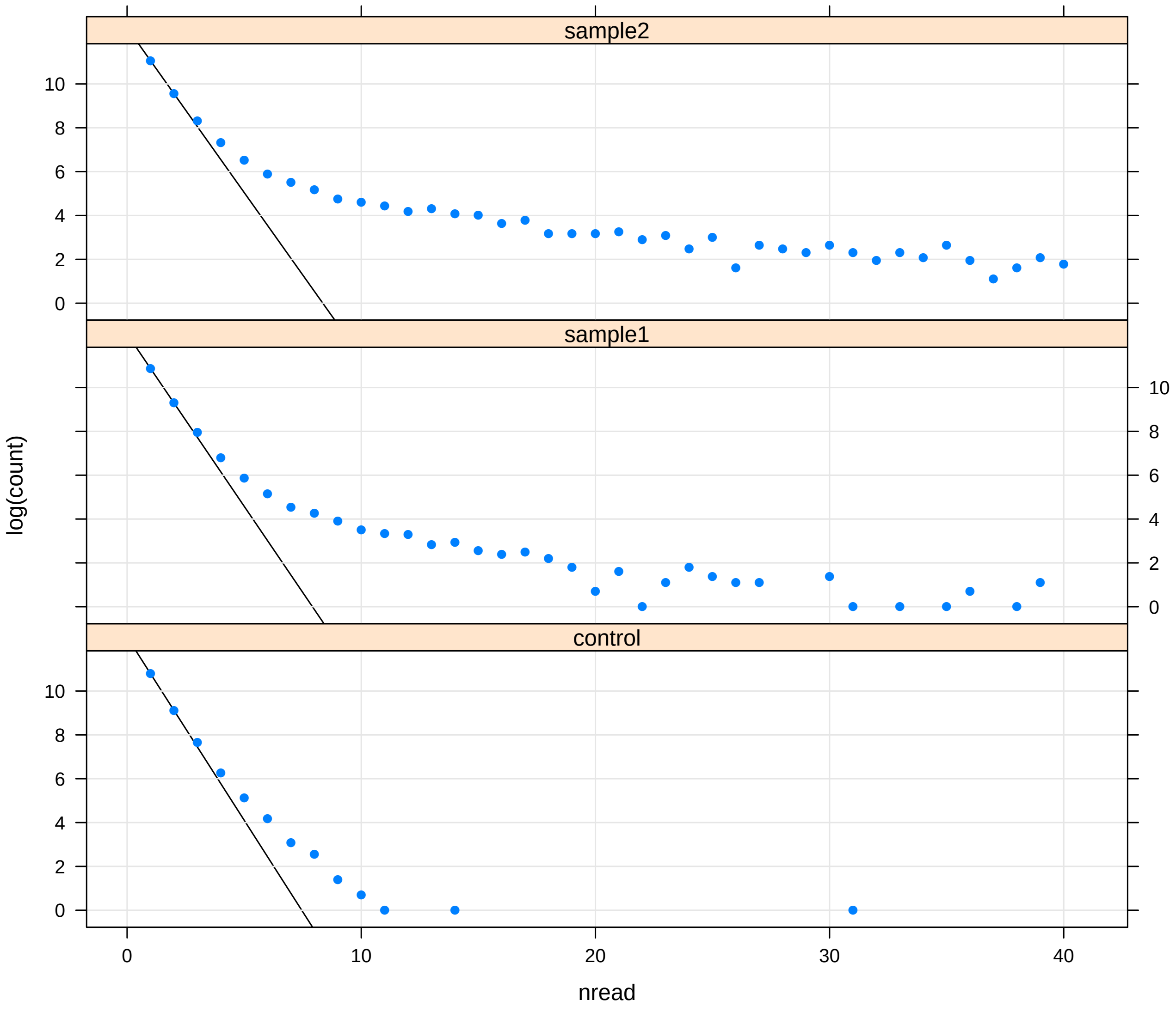


If reads were sampled randomly from the genome, then the null distribution number of reads per island would have a geometric distribution; that is,

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

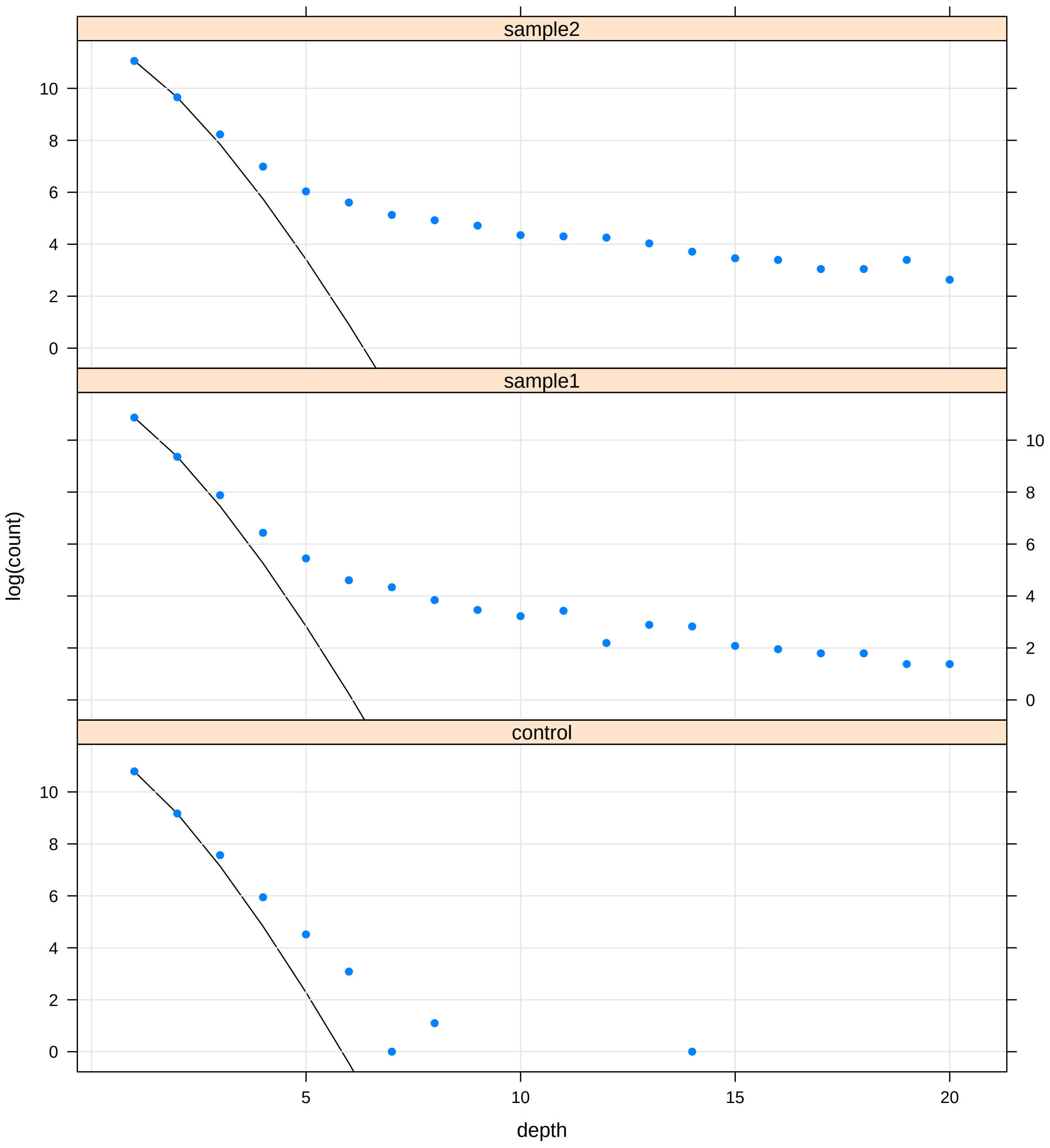
In other words, $\log P(X = k)$ is linear in k. Although our samples are not random, the islands with just one or two reads may be representative of the null distribution.

```
> xyplot(log(count) ~ nread | sample, nread.islands,
+        subset = (chromosome == "chr5" & nread <= 40),
+        layout = c(1, 3), 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, 1, max(end(g))), lower = 1)
+     tab <- table(viewMaxs(s))
+     ans <- data.frame(depth = as.numeric(names(tab)), count = as.numeric(tab))
+     ans
+ }
> depth.islands <- gdApply(alignedLocs, islandDepthSummary)
> depth.islands <- as(depth.islands, "data.frame")
> xyplot(log(count) ~ depth | sample, depth.islands,
+        subset = (chromosome == "chr5" & depth <= 20),
+        layout = c(1, 3), 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, ...)
+        })
```


sample2
sample1
control
log(count)
depth
0
2
4
6
8
10
5
10
15
20

Peaks

Going back to our example of chr5 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 152537259-length Rle subject

views:
          start       end width
  [1]   3802981   3803149   169 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
  [2]   3928831   3929037   207 [ 8  8  8  8  8  8  8  9  9  9  9  9  9  9  9]
  [3]   5365494   5365624   131 [8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9]
  [4]  12857038  12857134    97 [8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]
  [5]  15477731  15477940   210 [ 8  8  8  8  9  9 10 10 10 11 12 12 12 12 12]
  [6]  20650347  20650692   346 [ 8  8  9  9  9 10 10 10 10 10 10 10 10 10 10]
  [7]  22892864  22892977   114 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
  [8]  22924244  22924441   198 [ 8  8  8  8  8  8  8  8  8  8  8  8  9 10 10]
  [9]  22940028  22940156   129 [ 8  8  8  8  8  9  9  9  9  9 11 11 11 12 12]
  ...       ...       ...   ... ...
[268] 147606751 147606763    13 [8 8 8 8 8 8 8 8 8 8 8 8 8]
[269] 149419756 149420068   313 [ 8  8  8  8  8  9  9  9  9  9 10 10 10 10 10]
[270] 149770760 149770921   162 [8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]
[271] 149843042 149843332   291 [ 8  8  8  8  8  8  8  8  9  9  9  9  9  9  9]
[272] 150123241 150123329    89 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
[273] 150438937 150439070   134 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
[274] 151957422 151957494    73 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9]
[275] 151992582 151992783   202 [8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]
[276] 152101077 152101276   200 [ 9  9 10 10 10 10 10 11 11 11 11 11 11 11 11]
```

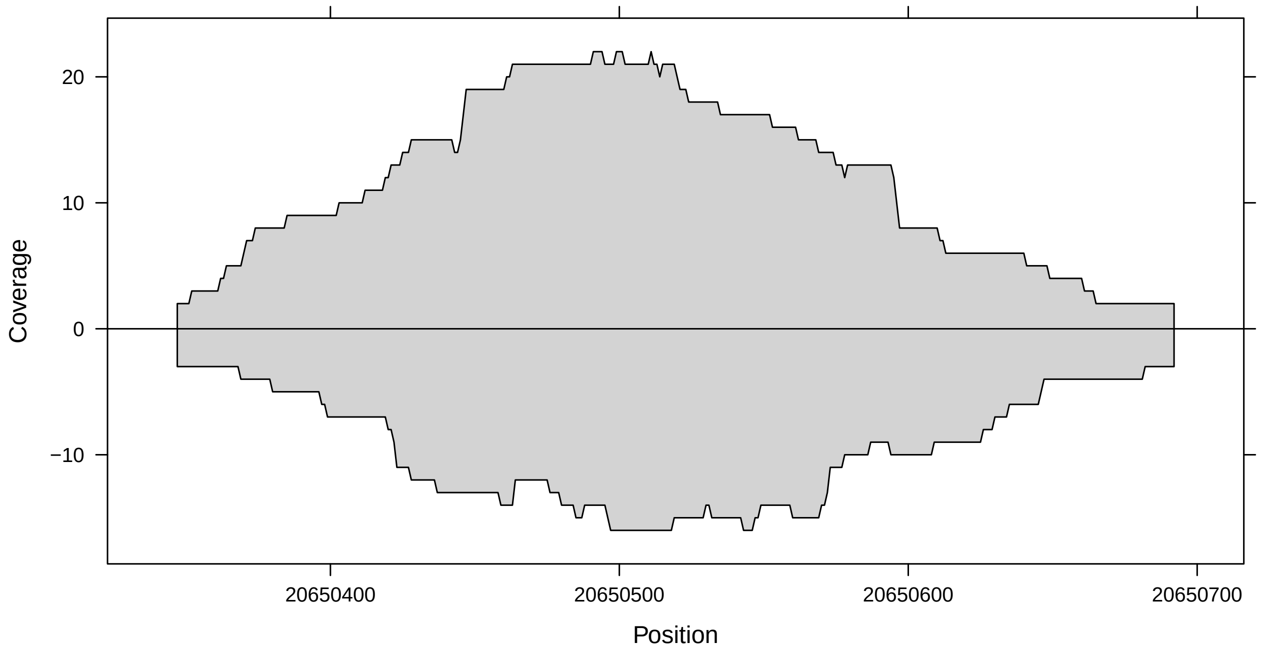
It is meaningful to ask about the contribution of each strand to each peak, as the sequenced region of pull-down fragments would be on opposite sides of a binding site depending on which strand it matched. We can compute strand-specific coverage, and look at the individual coverages under the combined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

```
> peak.depths <- viewMaxs(peaks)
> cov.pos <- coverage(extendReads(alignedLocs$sample1$chr5, strand = "+", seqLen = 150),
+                     start = 1, end = mouse.chromlens["chr5"])
> cov.neg <- coverage(extendReads(alignedLocs$sample1$chr5, strand = "-", seqLen = 150),
+                     start = 1, end = mouse.chromlens["chr5"])
> peaks.pos <- copyIRanges(peaks, cov.pos)
> peaks.neg <- copyIRanges(peaks, cov.neg)
> which(peak.depths >= 40)
```

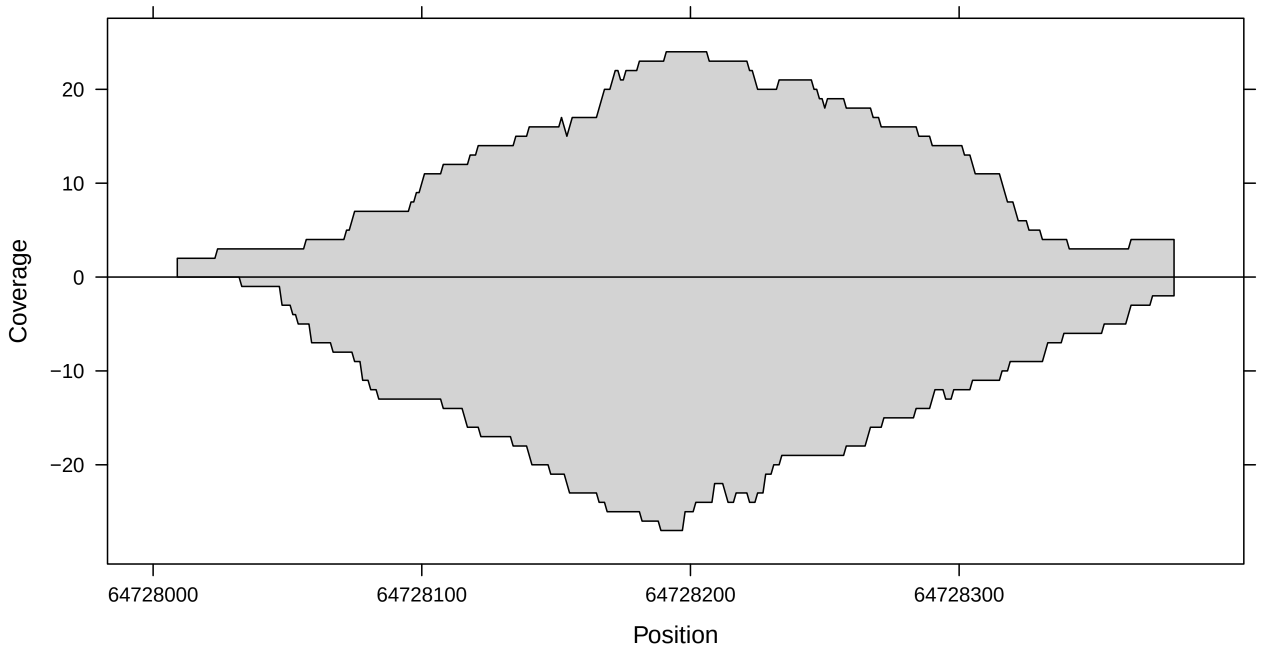
```
[1]   6  96 126 173
```

We plot the four highest peaks below.

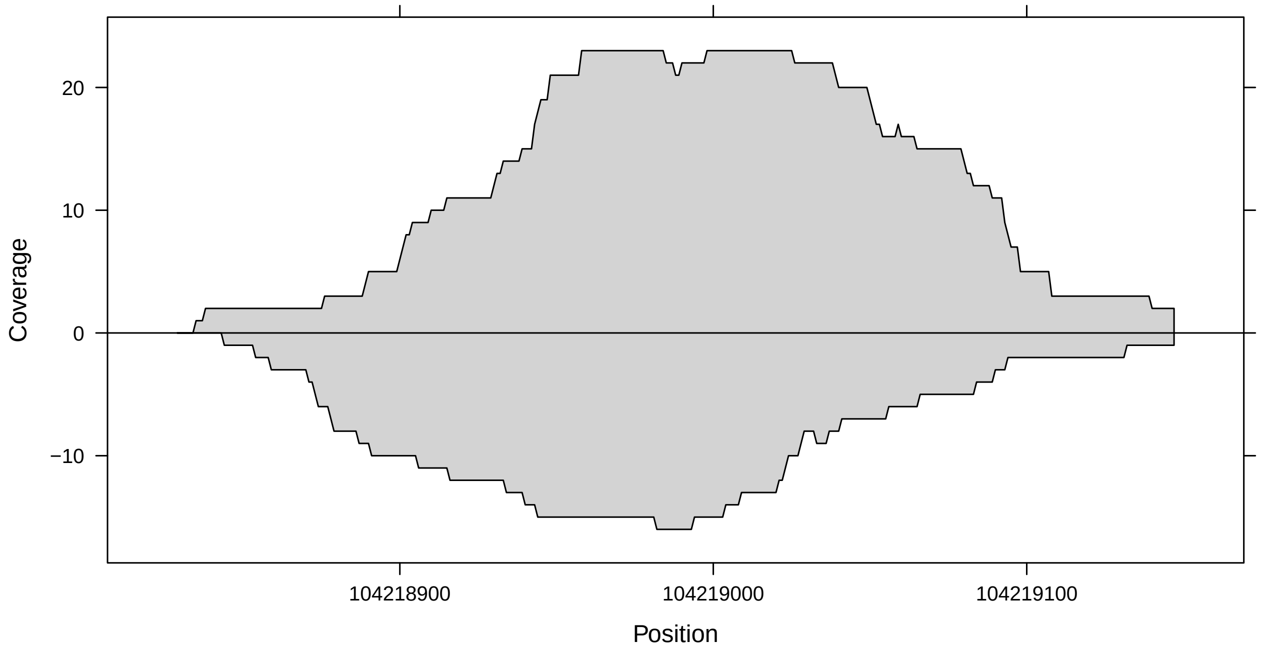
```
> plotPeak(peaks.pos[6], peaks.neg[6])
```



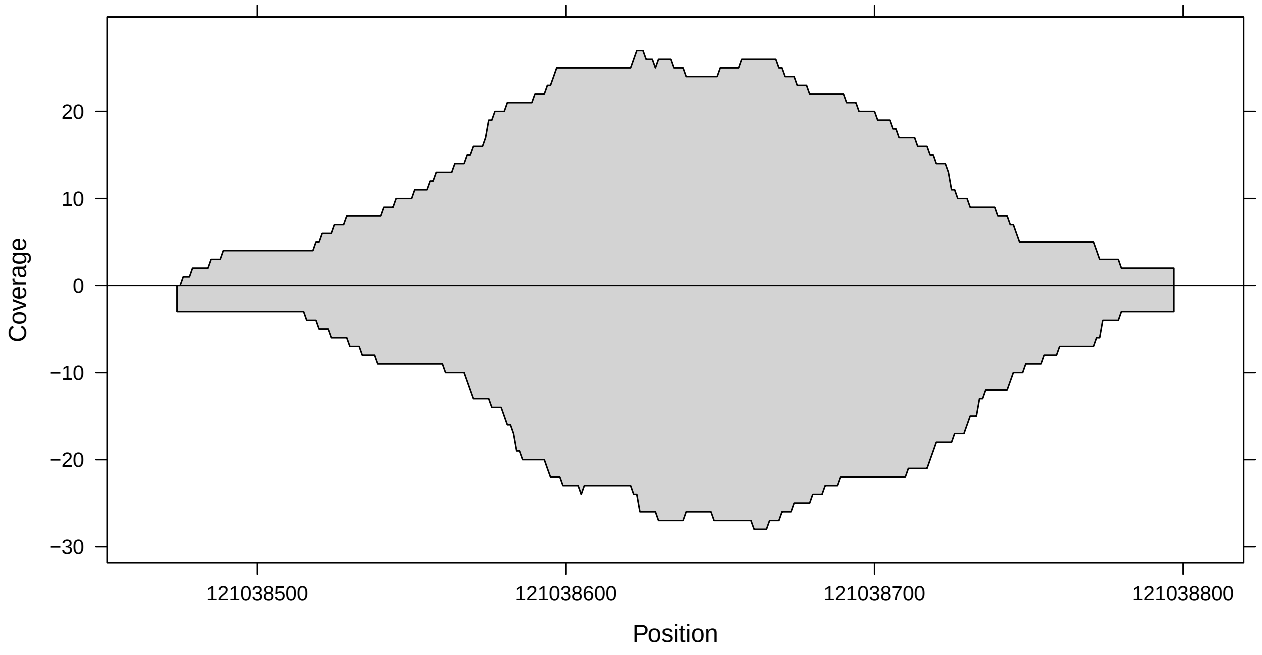
```
> plotPeak(peaks.pos[96], peaks.neg[96])
```



```
> plotPeak(peaks.pos[126], peaks.neg[126])
```



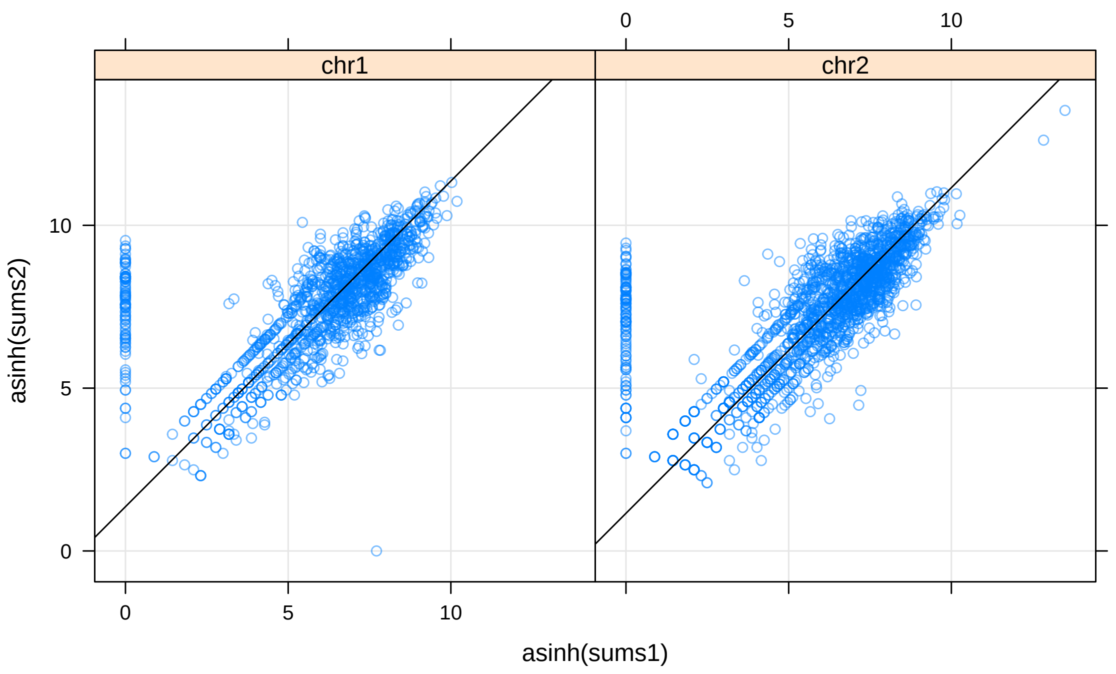
```
> plotPeak(peaks.pos[173], peaks.neg[173])
```



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(alignedLocs, extendReads, seqLen = 200)
> peakSummary <-
+     diffPeakSummary(extRanges$sample1, extRanges$sample2,
+                     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 sums1 sums2 maxs1 maxs2  down    up
1       chr1 6810057 6810265       18  1514  1635     9     9 FALSE FALSE
2       chr1 6810752 6810759       10     8    72     1     9 FALSE FALSE
```

```
3       chr1 6810772 6810929       14     0  1962     0    14 FALSE  TRUE
4       chr1 7078966 7078982       10    68   102     4     6 FALSE FALSE
5       chr1 7078992 7078996       10    20    30     4     6 FALSE FALSE
6       chr1 7079000 7079158       13   549  1187     4     9 FALSE FALSE
      resids      diffs
1 -1.2890212 0.07688763
2  0.8215419 2.19338924
3  6.8859561 8.27486689
4 -0.9613956 0.40543508
5 -0.9617114 0.40511836
6 -0.5967713 0.77108530
```

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` dataset in the `chipseq` package contains gene location information from UCSC. The `genomic_regions` function converts this into a set of ranges defining promoters, upstream regions, etc.

```
> data(geneMouse)
> gregions <- genomic_regions(genes = geneMouse, proximal = 2000)
> gregions$gene <- as.character(gregions$gene)
> str(gregions)
```

```
'data.frame':        49409 obs. of  12 variables:
 $ chrom           : Factor w/ 33 levels "chr1","chr10",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ gene            : chr  "uc007aet.1" "uc007aeu.1" "uc007aev.1" "uc007aew.1" ...
 $ promoter.start  : int  3203713 3659579 3646985 4397322 4348473 4481816 4484494 4484494 4484494 4484494 .
 $ promoter.end    : int  3207713 3663579 3650985 4401322 4352473 4485816 4488494 4488494 4488494 4488494 .
 $ threeprime.start: int  3193984 3202562 3636391 4278926 4332223 4479008 4479008 4479008 4479008 4479008 .
 $ threeprime.end  : int  3197984 3206562 3640391 4282926 4336223 4483008 4483008 4483008 4483008 4483008 .
 $ upstream.start  : int  3207714 3663580 3650986 4401323 4352474 4485817 4488495 4488495 4488495 4488495 .
 $ upstream.end    : int  3215713 3671579 3658985 4409322 4360473 4493816 4496494 4496494 4496494 4496494 .
 $ downstream.start: int  3185984 3194562 3628391 4270926 4324223 4471008 4471008 4471008 4471008 4471008 .
 $ downstream.end  : int  3193983 3202561 3636390 4278925 4332222 4479007 4479007 4479007 4479007 4479007 .
 $ gene.start      : int  3195984 3204562 3638391 4280926 4334223 4481008 4481008 4481008 4481008 4481008 .
 $ gene.end        : int  3205713 3661579 3648985 4399322 4350473 4483816 4486494 4486494 4486494 4486494 .
```

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

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

```
  type total promoter threeprime upstream downstream gene chromosome
1  all  1257      242         96      163        199  576       chr1
2   up    97        6          6       14         12   38       chr1
3 down    23        3          2        2          2   10       chr1
4  all  1034      261        101      188        187  548      chr10
5   up    77        5          8        7         12   36      chr10
6 down    16        4          0        1          1    5      chr10
```

```
> 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      18837 1650  277 0.168
promoter    4139  183   77 0.421
threeprime  1783  124   29 0.234
upstream    3200  218   38 0.174
downstream  3340  220   53 0.241
gene        9822  832  119 0.143
```

Version information

```
> sessionInfo()
```

```
R version 2.9.0 Under development (unstable) (2008-12-22 r47305)
x86_64-unknown-linux-gnu

locale:
LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF-8;LC_MONETARY=C;LC_MESSAGES=en_U

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

other attached packages:
[1] BSgenome.Mmusculus.UCSC.mm9_1.3.11 chipseq_0.1.8
[3] ShortRead_1.1.34                   lattice_0.17-17
[5] Biobase_2.3.9                      BSgenome_1.11.8
[7] Biostrings_2.11.22                 IRanges_1.1.33

loaded via a namespace (and not attached):
[1] grid_2.9.0         Matrix_0.999375-17 tools_2.9.0
```