

BANCseq KdApp Determination

Hannah Neikes

2023-06-07

This script is an example on how to determine absolute apparent binding affinities (K_d^{Apps}) in native chromatin by sequencing (BANC-seq).

```
library(foreach, quietly = T)
library(doParallel, quietly = T)
library(ComplexHeatmap, quietly = T)
library(circlize, quietly = T)
library(RColorBrewer, quietly = T)
library(ggplot2, quietly = T)
```

1. Load the data

Load a count table of raw counts for each sample at each peak location, as well as information on spike-in reads in each sample.

```
mm10 <- read.csv("E:/PhD/00_papers/10_BANCseq_Protocol/mm10-HN0059_1_1000nM_YY1_mESC_peaks.counts",
  stringsAsFactors = F, header = T, sep = "\t", skip = 1)
colnames(mm10)[7:ncol(mm10)] <- gsub("results_mouse.final_bam.mm10.HN0059_[1-9]{1}_([0-9]{4})nM_YY1_mESC",
  "\\1_nM_YY1", colnames(mm10)[7:ncol(mm10)])
# make sure the column names in this df match
# with samples names in the spikeIn txt
spikeIn <- read.csv("readCounts.txt", stringsAsFactors = F,
  header = T, sep = "\t")
spikeIn$spikeIn_yeast <- 1e+06/spikeIn$spikeInReads
spikeIn$sample <- gsub("HN0059_[1-9]{1}_([0-9]{4})nM_YY1_mESC",
  "\\1_nM_YY1", spikeIn$sample)
print(paste("Number of sites before KdApp determination:",
  nrow(mm10)))
```

```
## [1] "Number of sites before KdApp determination: 22472"
```

```
print(paste("Number of tested concentrations:", nrow(spikeIn)))
```

```
## [1] "Number of tested concentrations: 9"
```

```
head(mm10)
```

```
##      Geneid Chr   Start      End Strand Length 1000_nM_YY1 0564_nM_YY1 0500_nM_YY1
## 1         1 chr1 3670975 3671509      +    535         64         76         55
## 2         2 chr1 3671277 3671811      +    535         77         61         54
## 3         3 chr1 3671727 3672261      +    535         62         47         51
## 4         4 chr1 3672072 3672606      +    535         58         40         46
## 5         5 chr1 3681429 3681963      +    535         29         15         14
## 6         6 chr1 3852231 3852765      +    535         18         15         17
##      0250_nM_YY1 0125_nM_YY1 0050_nM_YY1 0010_nM_YY1 0001_nM_YY1 0000_nM_YY1
## 1              51          21          18          14          10          13
## 2              43          26          23          14          11          11
## 3              29          17          19           9           8          14
## 4              31          13          19           5           8          11
## 5              12           5          15           6          11           6
## 6              15          12          17           1           3           3
```

2. Normalisation

The following code normalizes reads in each sample to the spike-in DNA, and subsequently calculates the relative binding per tested concentration and site.

```
mm10_kd <- mm10
for (s in spikeIn$sample) {
  mm10_kd[, s] <- mm10[, s] * spikeIn$spikeIn_yeast[spikeIn$sample ==
    s]
}
# fold change of highest concentration over 0 nM
# sample
mm10_kd$fcOverControl <- log2((mm10_kd$`1000_nM_YY1` +
  2 * sd(mm10_kd$`0000_nM_YY1`))/(mm10_kd$`0000_nM_YY1` +
  2 * sd(mm10_kd$`0000_nM_YY1`)))
mm10_kd[, grep("nM", colnames(mm10_kd))] <- (mm10_kd[,
  grep("nM", colnames(mm10_kd))] + 1)/(mm10_kd$`0000_nM_YY1` +
  1)
# relative signal at each binding site for each
# concentration
mm10_kd[, grep("nM", colnames(mm10_kd))] <- mm10_kd[,
  grep("nM", colnames(mm10_kd))]/apply(mm10_kd[,
  grep("nM", colnames(mm10_kd))], 1, max)
head(mm10_kd)
```

```
##      Geneid Chr   Start      End Strand Length 1000_nM_YY1 0564_nM_YY1 0500_nM_YY1
## 1         1 chr1 3670975 3671509      +    535    0.966694    1.000000    0.6042279
## 2         2 chr1 3671277 3671811      +    535    1.000000    0.6902142    0.5101113
## 3         3 chr1 3671727 3672261      +    535    1.000000    0.6605065    0.5983055
## 4         4 chr1 3672072 3672606      +    535    1.000000    0.6009474    0.5768908
## 5         5 chr1 3681429 3681963      +    535    1.000000    0.4510472    0.3515890
## 6         6 chr1 3852231 3852765      +    535    1.000000    0.7263133    0.6871863
##      0250_nM_YY1 0125_nM_YY1 0050_nM_YY1 0010_nM_YY1 0001_nM_YY1 0000_nM_YY1
## 1    0.6375952    0.2731234    0.2525032    0.30983136    0.08608027    0.13522254
```

```
## 2 0.4622941 0.2906887 0.2773400 0.26641188 0.08138702 0.09843332
## 3 0.3873089 0.2361677 0.2845827 0.21282426 0.07361272 0.15546870
## 4 0.4425319 0.1931478 0.3042008 0.12657373 0.07868732 0.13066514
## 5 0.3429773 0.1490310 0.4802922 0.30348052 0.21598310 0.14286665
## 6 0.6900237 0.5740578 0.8763458 0.08257862 0.09583937 0.11570606
## fcdOverControl
## 1 0.4006316
## 2 0.4925038
## 3 0.3821209
## 4 0.3735736
## 5 0.2000780
## 6 0.1329035
```

3. K_d^{APP} Determination

This is the actual K_d^{APP} determination, based on (log10-transformed) concentrations (xValues) and relative binding per peak (yValues).

```
mm10_kd$kd <- NA
mm10_kd$p <- NA
mm10_kd$r <- NA
mm10_kd$n <- NA
xValues <- c(log10(as.numeric(gsub("[0-9]{4}_nM_YY1",
  "\\1", colnames(mm10_kd[, grep("*nM", colnames(mm10_kd))])))))
xValues <- xValues[xValues >= 0]

cl <- makeCluster(10)
registerDoParallel(cl)
ptm <- proc.time()
mm10_kd <- foreach(i = 1:nrow(mm10_kd), .combine = rbind,
  .errorhandling = "remove") %dopar% {
  library(minpack.lm)
  mm10_kdTemp <- mm10_kd[i, ]
  yValues <- c(as.numeric(mm10_kd[i, grep("*nM",
    colnames(mm10_kd[, colnames(mm10_kd) != "0000_nM_YY1"])]))) # Don't use the OnM value!
  myModel <- nlsLM(yValues ~ 1/(((kd/xValues)^n) +
    1), start = list(kd = 1, n = 1), control = nls.lm.control(maxiter = 50))
  myCoefs <- coef(myModel)
  myCor <- cor.test(yValues, predict(myModel))
  mm10_kdTemp$kd <- 10^(myCoefs[[1]])
  mm10_kdTemp$p <- myCor$p.value
  mm10_kdTemp$r <- myCor$estimate[[1]]
  mm10_kdTemp$n <- myCoefs[[2]]
  mm10_kdTemp
}
stopCluster(cl)
head(mm10_kd)
```

```
## Geneid Chr Start End Strand Length 1000_nM_YY1 0564_nM_YY1 0500_nM_YY1
## 1 1 chr1 3670975 3671509 + 535 0.966694 1.0000000 0.6042279
## 2 2 chr1 3671277 3671811 + 535 1.000000 0.6902142 0.5101113
## 3 3 chr1 3671727 3672261 + 535 1.000000 0.6605065 0.5983055
```

```
## 4      4 chr1 3672072 3672606      +      535      1.000000      0.6009474      0.5768908
## 5      5 chr1 3681429 3681963      +      535      1.000000      0.4510472      0.3515890
## 6      6 chr1 3852231 3852765      +      535      1.000000      0.7263133      0.6871863
##      0250_nM_YY1 0125_nM_YY1 0050_nM_YY1 0010_nM_YY1 0001_nM_YY1 0000_nM_YY1
## 1      0.6375952      0.2731234      0.2525032      0.30983136      0.08608027      0.13522254
## 2      0.4622941      0.2906887      0.2773400      0.26641188      0.08138702      0.09843332
## 3      0.3873089      0.2361677      0.2845827      0.21282426      0.07361272      0.15546870
## 4      0.4425319      0.1931478      0.3042008      0.12657373      0.07868732      0.13066514
## 5      0.3429773      0.1490310      0.4802922      0.30348052      0.21598310      0.14286665
## 6      0.6900237      0.5740578      0.8763458      0.08257862      0.09583937      0.11570606
##      fcOverControl      kd      p      r      n
## 1      0.4006316 174.36680 0.001891538 0.9066907 7.075834
## 2      0.4925038 225.38452 0.003841540 0.8810458 4.727067
## 3      0.3821209 272.16071 0.001310139 0.9176756 6.499369
## 4      0.3735736 279.05862 0.001364914 0.9165191 6.194282
## 5      0.2000780 383.61441 0.207707046 0.4993598 1.537376
## 6      0.1329035 36.16809 0.005221303 0.8677758 2.670069
```

4. Select high confidence sites

Select sites with high confidence K_d^{APP} fit (based on r- and p-value), and remove outliers. For downstream analysis, we also save the results.

```
mm10_kd$highConf <- F
mm10_kd$highConf[mm10_kd$p < 0.01 & mm10_kd$r > 0.9] <- T
mm10_kd$outlierMeanSd_n <- F
mm10_kd$outlierMeanSd_kd <- F
my_mean_n <- mean(mm10_kd$n[mm10_kd$highConf])
my_sd_n <- sd(mm10_kd$n[mm10_kd$highConf])
my_up_n <- my_mean_n + (2 * my_sd_n) # Upper Range
my_low_n <- my_mean_n - (2 * my_sd_n) # Lower Range
mm10_kd$outlierMeanSd_n[(mm10_kd$n > my_up_n | mm10_kd$n <
  my_low_n)] <- T
my_mean_kd <- mean(mm10_kd$kd[mm10_kd$highConf])
my_sd_kd <- sd(mm10_kd$kd[mm10_kd$highConf])
my_up_kd <- my_mean_kd + (2 * my_sd_kd) # Upper Range
my_low_kd <- my_mean_kd - (2 * my_sd_kd) # Lower Range
mm10_kd$outlierMeanSd_kd[(mm10_kd$kd > my_up_kd | mm10_kd$kd <
  my_low_kd)] <- T
mm10_kd$outlierMeanSd <- F
mm10_kd$outlierMeanSd[mm10_kd$outlierMeanSd_n == T &
  mm10_kd$outlierMeanSd_kd != T] <- "n"
mm10_kd$outlierMeanSd[mm10_kd$outlierMeanSd_n != T &
  mm10_kd$outlierMeanSd_kd == T] <- "kd"
mm10_kd$outlierMeanSd[mm10_kd$outlierMeanSd_n == T &
  mm10_kd$outlierMeanSd_kd == T] <- "n_kd"
rm(my_low_kd, my_up_kd, my_sd_kd, my_mean_kd, my_mean_n,
  my_low_n, my_up_n, my_sd_n)

mm10_kd_highConf <- mm10_kd[mm10_kd$highConf == T &
  mm10_kd$outlierMeanSd != "n_kd", ]
write.table(mm10_kd_highConf, file = "highConfPeaks_noOutliers.txt",
```

```
quote = F, col.names = T, row.names = F, sep = "\t")
print(paste("Number of high confidence sites:", nrow(mm10_kd_highConf)))
```

```
## [1] "Number of high confidence sites: 10206"
```

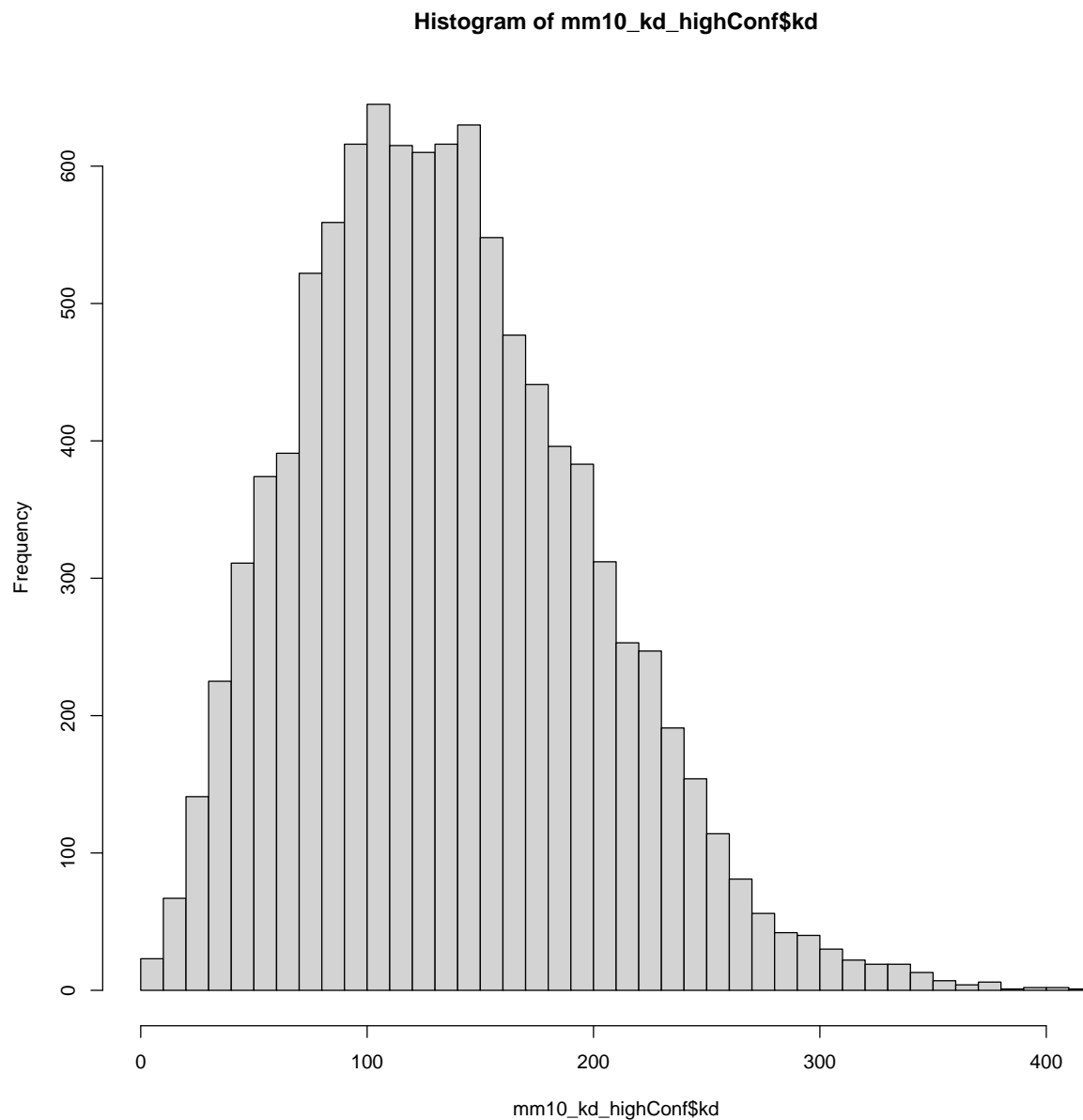
```
head(mm10_kd_highConf)
```

```
##      Geneid Chr   Start      End Strand Length 1000_nM_YY1 0564_nM_YY1
## 1         1 chr1 3670975 3671509      +    535   0.9666940   1.0000000
## 3         3 chr1 3671727 3672261      +    535   1.0000000   0.6605065
## 4         4 chr1 3672072 3672606      +    535   1.0000000   0.6009474
## 8         8 chr1 4492613 4493147      +    535   0.9316848   1.0000000
## 9         9 chr1 4493265 4493799      +    535   1.0000000   0.7658913
## 13        13 chr1 4807752 4808286      +    535   1.0000000   0.9322908
##      0500_nM_YY1 0250_nM_YY1 0125_nM_YY1 0050_nM_YY1 0010_nM_YY1 0001_nM_YY1
## 1      0.6042279   0.6375952   0.2731234   0.2525032   0.30983136  0.08608027
## 3      0.5983055   0.3873089   0.2361677   0.2845827   0.21282426  0.07361272
## 4      0.5768908   0.4425319   0.1931478   0.3042008   0.12657373  0.07868732
## 8      0.7501499   0.6472058   0.2435986   0.3089649   0.31690474  0.13257066
## 9      0.7511983   0.7366573   0.2837811   0.3467953   0.06460228  0.10628787
## 13     0.7145244   0.4938274   0.4529814   0.2932666   0.07744753  0.15970566
##      0000_nM_YY1 fcOverControl      kd      p      r      n highConf
## 1      0.1352225      0.4006316 174.3668 0.0018915380 0.9066907 7.075834    TRUE
## 3      0.1554687      0.3821209 272.1607 0.0013101391 0.9176756 6.499369    TRUE
## 4      0.1306651      0.3735736 279.0586 0.0013649143 0.9165191 6.194282    TRUE
## 8      0.1489352      0.3488962 163.4329 0.0012136108 0.9197929 7.383073    TRUE
## 9      0.0832938      0.5742608 138.7901 0.0002744890 0.9515267 5.564925    TRUE
## 13     0.2293404      0.3213060 140.2030 0.0002981419 0.9501551 5.604257    TRUE
##      outlierMeanSd_n outlierMeanSd_kd outlierMeanSd
## 1              FALSE              FALSE          FALSE
## 3              FALSE              TRUE           kd
## 4              FALSE              TRUE           kd
## 8              FALSE              FALSE          FALSE
## 9              FALSE              FALSE          FALSE
## 13             FALSE              FALSE          FALSE
```

5. Plot the data

To have a quick glance at the K_d^{APPS} and relative enrichment, we can plot this histogram of K_d^{APPS} , as well as a Heatmap of the relative enrichment for each site and concentration, alongside the K_d^{APP} . The data frame consists of the peak information, relative signal at each site and concentration, as well as K_d^{APPS} and p- and r-values for the Hill curve fit.

```
hist(mm10_kd_highConf$kd, breaks = 50)
```



```
mm10_kd_highConf <- mm10_kd_highConf[order(-mm10_kd_highConf$kd),
] # sort the sites by KdApp for visualisation
Heatmap(mm10_kd_highConf[, 15:7], cluster_rows = F,
cluster_columns = F, name = "Relative signal\nat peak",
width = unit(5, "cm"), top_annotation = HeatmapAnnotation(`Flag-YY1 (nM)` = anno_barplot(as.numeric
"\1", colnames(mm10_kd_highConf[, 15:7]))),
bar_width = 0.75, border = F, axis_param = list(side = "left",
at = c(0, 500, 1000), labels = c(0, 500,
1000))), height = unit(1.5, "cm")),
col = colorRamp2(breaks = c(0, 0.25, 0.5, 0.75,
1), colors = brewer.pal(5, "YlGnBu"))) + Heatmap(mm10_kd_highConf[,
"kd"], name = "KdApp", width = unit(0.5, "cm"),
col = colorRamp2(breaks = c(seq(min(mm10_kd_highConf$kd),
max(mm10_kd_highConf$kd), length.out = 5)),
```

```
brewer.pal(5, "Spectral"))
```

