Stats with R and RStudio

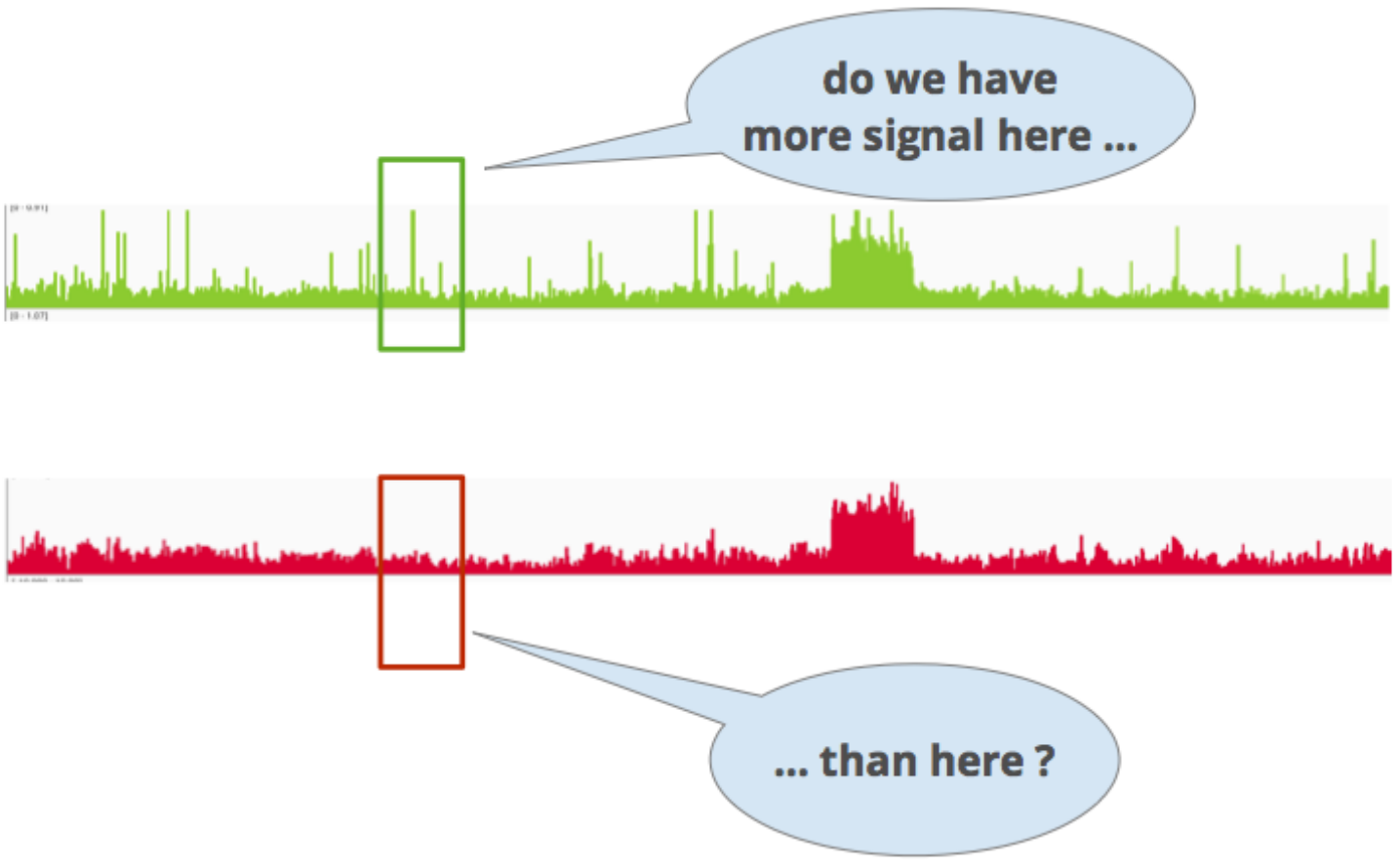
Practical: basic stats for peak calling

Jacques van Helden, Hugo varet and Julie Aubert

2016-11-23

Introduction

Peak-calling: question



Data loading

Defining the data directory

We will first define the URL from which the data can be downloaded, by concatenating the URL fo the course with the path to our dataset.
To concatenate paths, it is *recommended* to use the **R** command `file.path()`.

```
url.course <- "http://jvanheld.github.io/stats_avec_RStudio
url.data <- file.path(url.course, "practicals", "02_peak-ca
```

Loading a data table

R enables to download data directly from the Web.
Load counts per bin in chip sample.

```
## Define URL of the ChIP file
chip.bedg.file <- file.path(url.data, "FNR_200bp.bedg")

## Load the file content in an R data.frame
chip.bedg <- read.table(chip.bedg.file)

## Set column names
names(chip.bedg) <- c("chrom", "start", "end","counts")
```

Exploring a data frame: dim()

Before anything else, let us inspect the size of the data frame, in order to check that it was properly lodaded.

```
dim(chip.bedg)
```

```
## [1] 23199     4
```

Checking the n first rows: head()

The function `head()` displays the first rows of a table.

```
head(chip.bedg, n = 5)
```

```
##                                chrom start   end counts
## 1 gi|49175990|ref|NC_000913.2|     0   200   1594
## 2 gi|49175990|ref|NC_000913.2|   200   400    834
## 3 gi|49175990|ref|NC_000913.2|   400   600    222
## 4 gi|49175990|ref|NC_000913.2|   600   800    172
## 5 gi|49175990|ref|NC_000913.2|   800  1000    123
```

Checking the n last rows: tail()

The function `tail()` displays the last rows of a table.

```
tail(chip.bedg, n = 5)
```

```
##                                      chrom   start     end count
## 23195 gi|49175990|ref|NC_000913.2| 4638800 4639000    15
## 23196 gi|49175990|ref|NC_000913.2| 4639000 4639200     9
## 23197 gi|49175990|ref|NC_000913.2| 4639200 4639400     9
## 23198 gi|49175990|ref|NC_000913.2| 4639400 4639600    22
## 23199 gi|49175990|ref|NC_000913.2| 4639600 4639675    18
```

Viewing a table

The function `View()` displays the full table in a user-friendly mode.

```
View(chip.bedg)
```

Selecting arbitrary rows

```
chip.bedg[100:105,]
```

```
##                                    chrom start   end counts
## 100 gi|49175990|ref|NC_000913.2| 19800 20000    21
## 101 gi|49175990|ref|NC_000913.2| 20000 20200     0
## 102 gi|49175990|ref|NC_000913.2| 20200 20400     0
## 103 gi|49175990|ref|NC_000913.2| 20400 20600   108
## 104 gi|49175990|ref|NC_000913.2| 20600 20800   229
## 105 gi|49175990|ref|NC_000913.2| 20800 21000   245
```

Selecting arbitrary columns

```
chip.bedg[100:105, 2]
```

```
## [1] 19800 20000 20200 20400 20600 20800
```

```
chip.bedg[100:105, "start"]
```

```
## [1] 19800 20000 20200 20400 20600 20800
```

```
chip.bedg[100:105, c("start", "counts")]
```

```
##      start counts
## 100 19800     21
## 101 20000      0
## 102 20200      0
## 103 20400    108
## 104 20600    229
## 105 20800    245
```

Adding columns

We can add columns with the result of computations from other columns.

```
chip.bedg$midpos <- (chip.bedg$start + chip.bedg$end)/2
head(chip.bedg)
```

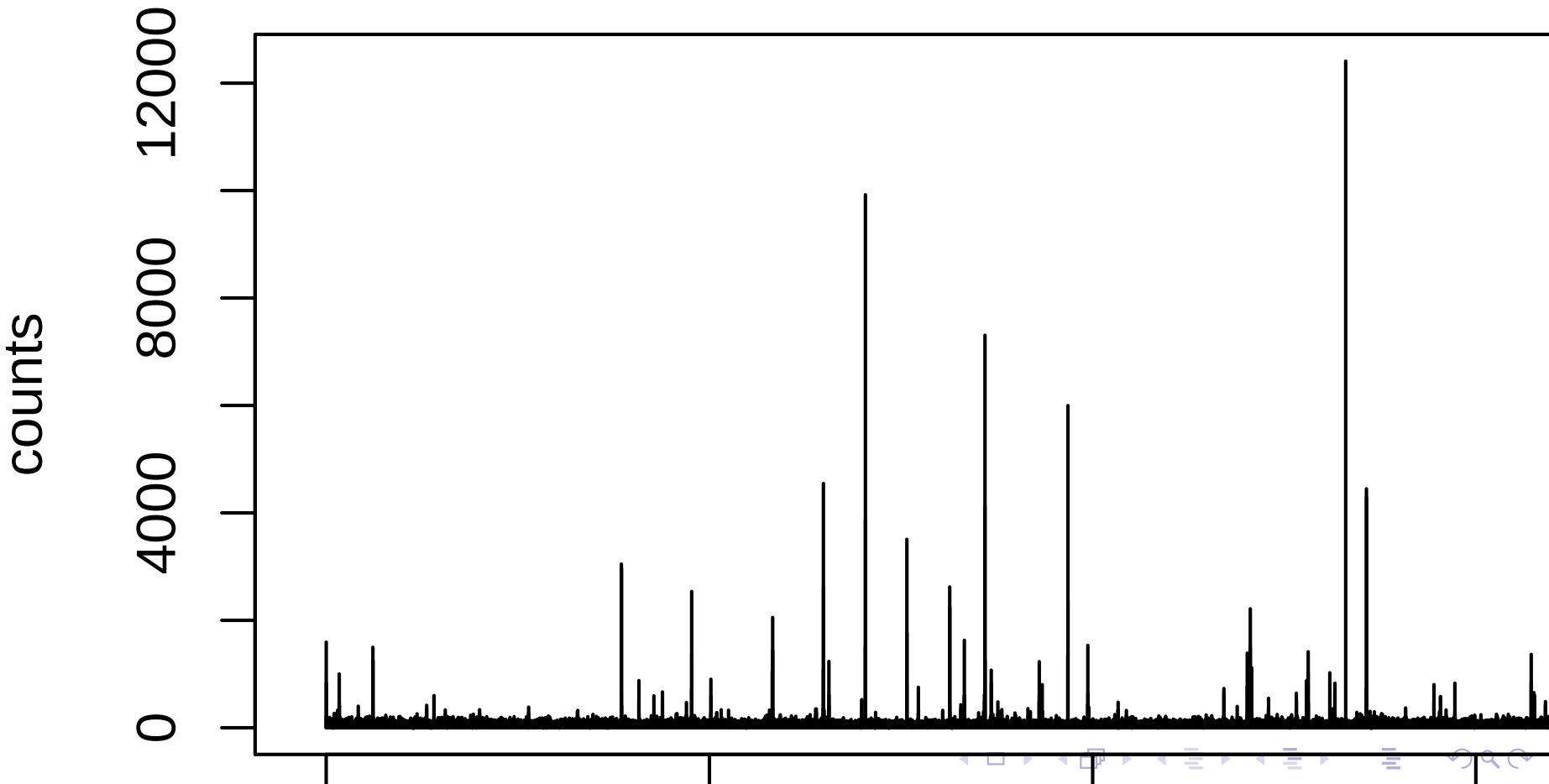
```
##                                  chrom start  end counts midpos
## 1 gi|49175990|ref|NC_000913.2|     0   200 1594    100
## 2 gi|49175990|ref|NC_000913.2|   200   400  834    300
## 3 gi|49175990|ref|NC_000913.2|   400   600  222    500
## 4 gi|49175990|ref|NC_000913.2|   600   800  172    700
## 5 gi|49175990|ref|NC_000913.2|   800  1000  123    900
## 6 gi|49175990|ref|NC_000913.2|  1000  1200  116   1100
```

Exploring the data with basic plots

Plotting a density profile

We can readily print a plot with the counts per bin.

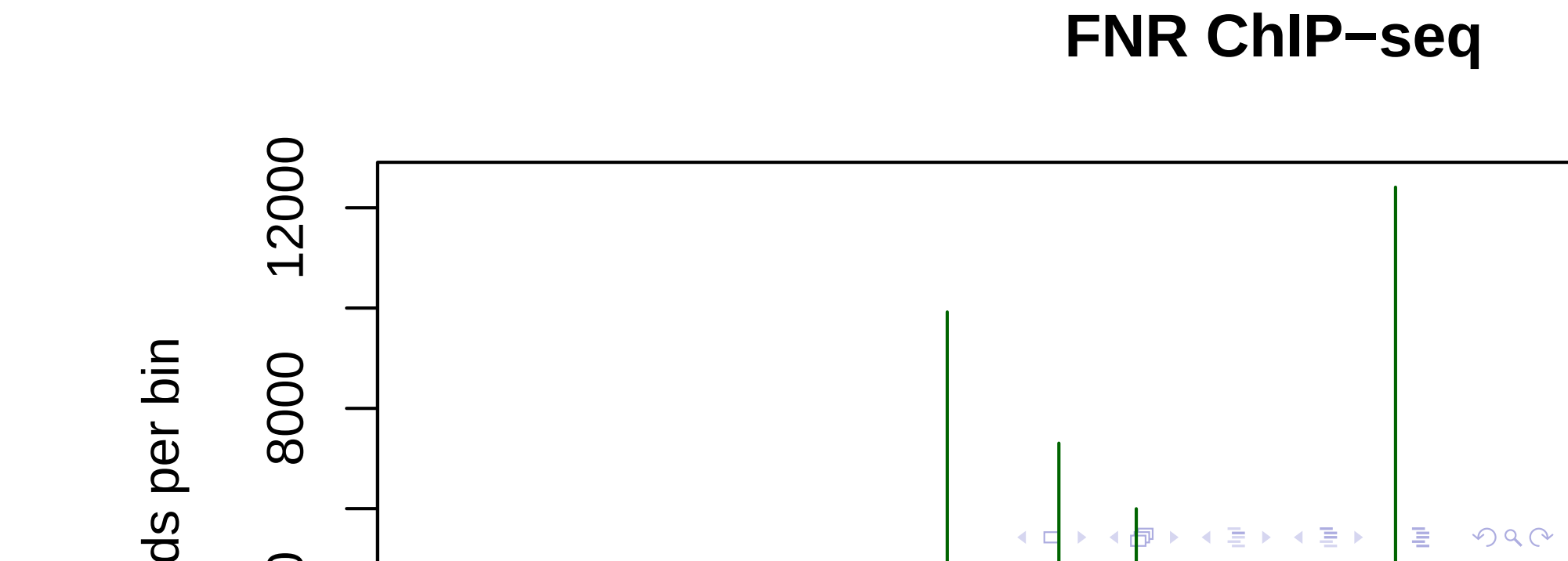
```
plot(chip.bedg[, c("midpos", "counts")], type="h")
```



Plotting a density profile

Let us improve the plot

```
plot(chip.bedg[, c("midpos", "counts")], type="h",
     col="darkgreen", xlab="Genomic position (200bp bins)",
     ylab= "Reads per bin",
     main="FNR ChIP-seq")
```



Exercise: exploring the background

We already loaded the count table for the FNR ChIP counts per bin. The background level will be estimated by loading counts per bin in a genomic input sample. These counts are available in the same directory a file named `input_200bp.bedg`

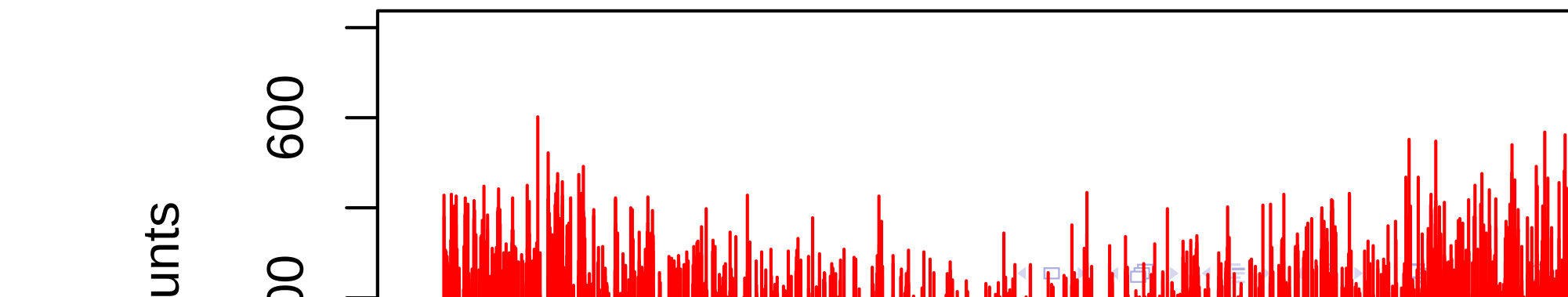
1. Load the counts per bin in the input sample (genome sequencing).
2. Plot the density profile of the input
3. Compare chip-seq and input density profiles
4. Compare counts per bin between chip-seq and input

Solution: loading the input counts per bin

```
## Define URL of the input file
input.bedg.file <- file.path(url.data, "input_200bp.bedg")

## Load the file content in an R data.frame
input.bedg <- read.table(input.bedg.file)

## Set column names
names(input.bedg) <- c("chrom", "start", "end","counts")
```

Solution: plotting the input density profile

```
## Compute middle positions per bin
input.bedg$midpos <- (input.bedg$start + input.bedg$end)/2

plot(input.bedg[, c("midpos", "counts")], type="h",
     col="red", xlab="Genomic position (200bp bins)",
     ylab= "Read counts",
     main="Background (genomic input)")
```



Solution: comparing chip-seq and background density profiles

```
par(mfrow=c(1,2)) ## Draw two panels besides each other
plot(chip.bedg[, c("midpos", "counts")], type="h",
     col="darkgreen", xlab="Genomic position (200bp bins)",
     ylab= "Reads per bin",
     main="FNR ChIP-seq")
plot(input.bedg[, c("midpos", "counts")], type="h",
     col="red", xlab="Genomic position (200bp bins)",
     ylab= "Reads per bin",
     main="Background (genomic input)")
```

Solution: comparing counts per bin between chip-seq and input

```
plot(input.bedg$counts, chip.bedg$counts, col="darkviolet",
     xlab="Genomic input", ylab="FNR ChIP-seq",
     main="Reads per 200bp bin")
```

Solution: comparing counts per bin between chip-seq and input

In order to better highight the dynamic range, we can use a log-based representation

```
plot(input.bedg$counts, chip.bedg$counts, col="darkviolet",
     xlab="Genomic input", ylab="FNR ChIP-seq",
     main="Reads per 200bp bin",
     log="xy")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 377 x v
## from logarithmic plot
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 403 y v
## from logarithmic plot
```

```
grid() ## add a grid
```

Questions

- On the ChIP-seq versus input plot, how would you define peaks ?
- Where would you place the limit between peaks and background fluctuations ?

Exercises

1. Think about further drawing modes to improve your perception of the differences between signal and background.
2. We will formulate (together) a reasoning path to compute a p-value for each peak.

Step-by-step walk to the significance

Merge the two tables

```
names(input.bedg)
```

```
## [1] "chrom"  "start"  "end"    "counts" "midpos"
```

```
## Merge two tables by identical values for multiple columns
count.table <- merge(chip.bedg, input.bedg, by=c("chrom", "
```

```
## Check the result size
names(count.table)
```

```
## [1] "chrom"         "start"          "end"            "midpos
## [5] "counts.chip"   "counts.input"
```

Checking the merge() result

```
## Simplify the chromosome name
head(count.table$chrom)
```

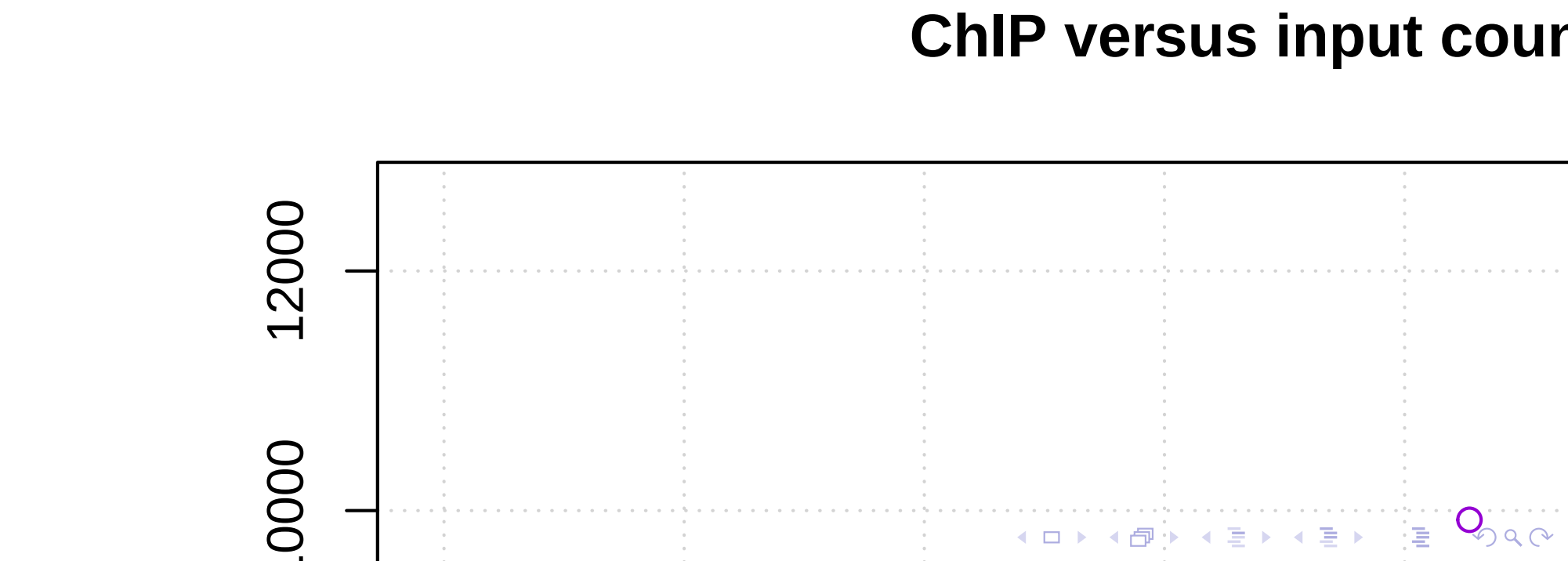
```
## [1] gi|49175990|ref|NC_000913.2| gi|49175990|ref|NC_000
## [3] gi|49175990|ref|NC_000913.2| gi|49175990|ref|NC_000
## [5] gi|49175990|ref|NC_000913.2| gi|49175990|ref|NC_000
## Levels: gi|49175990|ref|NC_000913.2|
```

```
count.table$chrom <- "NC_000913.2"
kable(head(count.table)) ## Display the head of the table
```

chrom	start	end	midpos	counts.chip	counts.inp
NC_000913.2	0	200	100	1594	5
NC_000913.2	1000	1200	1100	116	3
NC_000913.2	10000	10200	10100	116	3
NC_000913.2	100000	100200	100100	107	2

ChIP vs input counts per bin

```
max.counts <- max(count.table[, c("counts.input", "counts.c
plot(x = count.table$counts.input, xlab="Input counts",
     y = count.table$counts.chip, ylab="ChIP counts",
     main="ChIP versus input counts",
     col="darkviolet", panel.first=grid())
```



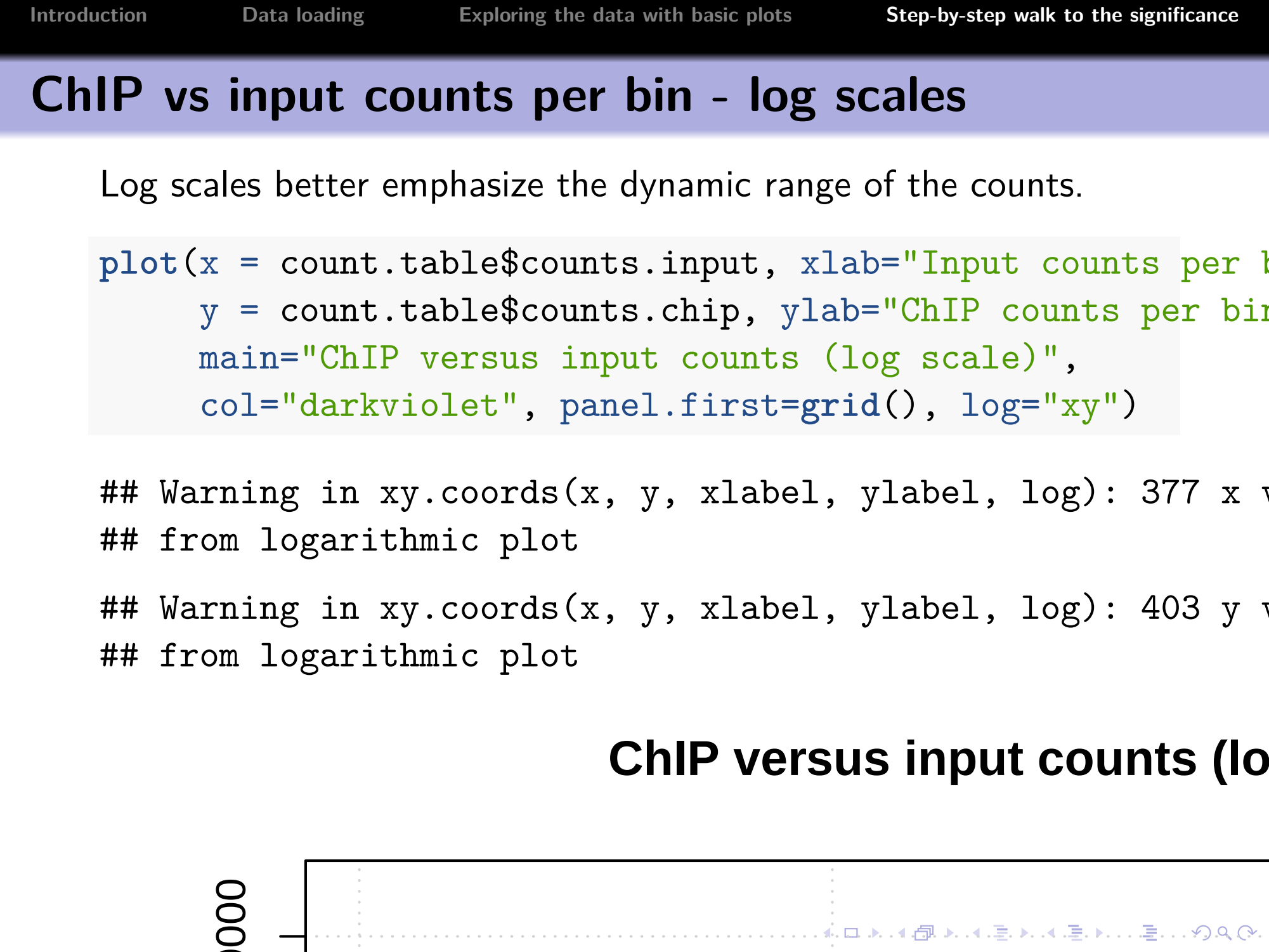
ChIP vs input counts per bin - log scales

Log scales better emphasize the dynamic range of the counts.

```
plot(x = count.table$counts.input, xlab="Input counts per b
     y = count.table$counts.chip, ylab="ChIP counts per bin
     main="ChIP versus input counts (log scale)",
     col="darkviolet", panel.first=grid(), log="xy")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 377 x v
## from logarithmic plot
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 403 y v
## from logarithmic plot
```



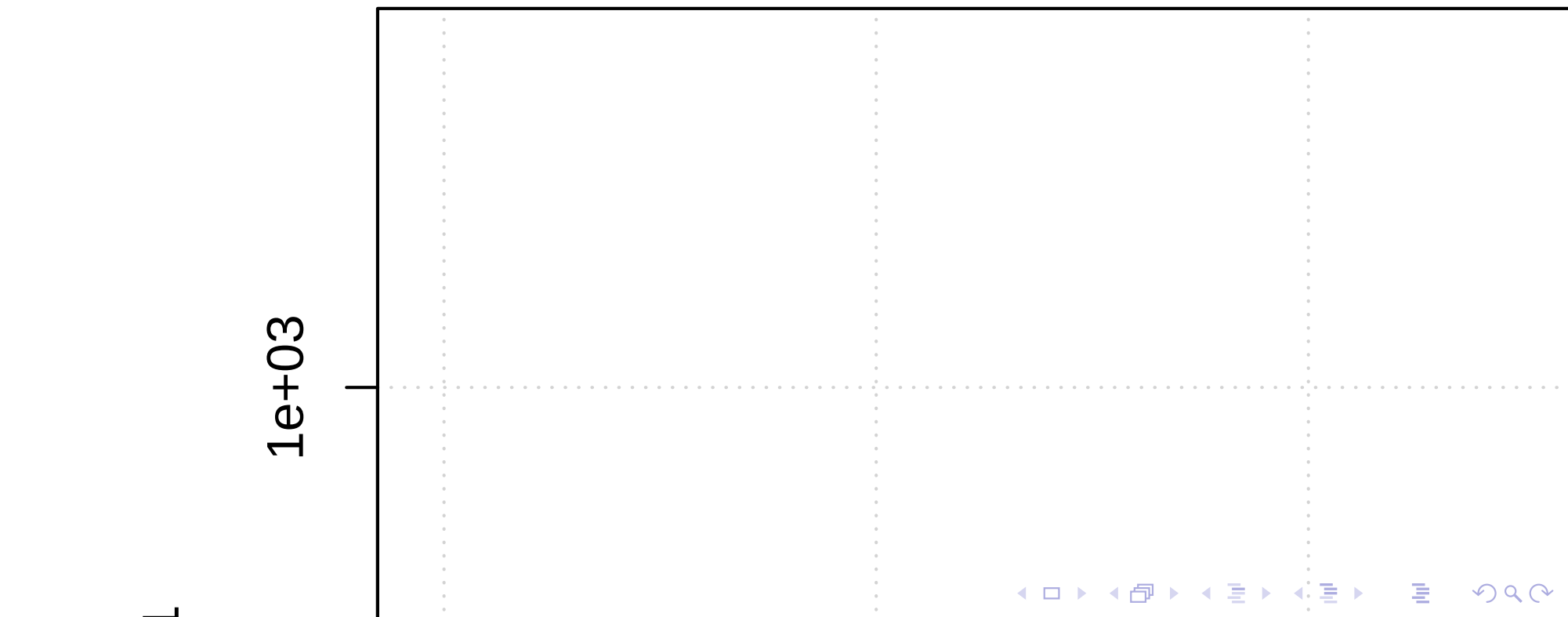
A tricky way to treat 0 values on log scales

- We can add a pseudo-count to avoid -Inf with 0 values.
- The usual pseudo-count is 1. I prefer 0.01 (this could be negotiated).

```
epsilon <- 0.1 ## Define a small pseudo-count
plot(x = count.table$counts.input + epsilon,
     y = count.table$counts.chip + epsilon,
     col="darkviolet", panel.first=grid(), log="xy")
```

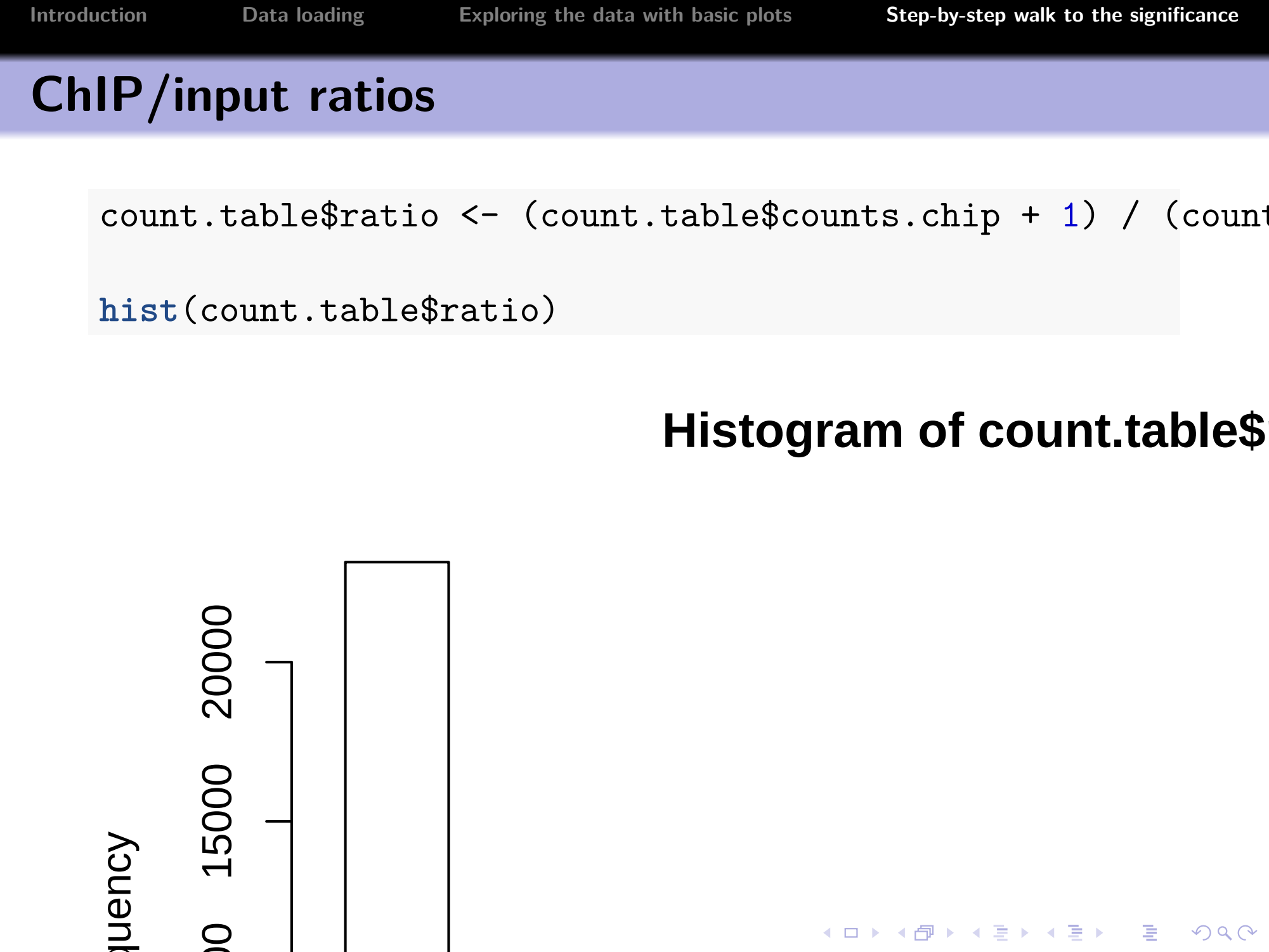
Drawing a diagonal

```
plot(x = count.table$counts.input + epsilon, xlab=paste("In
     y = count.table$counts.chip + epsilon,  ylab=paste("Ch
     col="darkviolet", panel.first=grid(), log="xy")
abline(a=0, b=1)
```



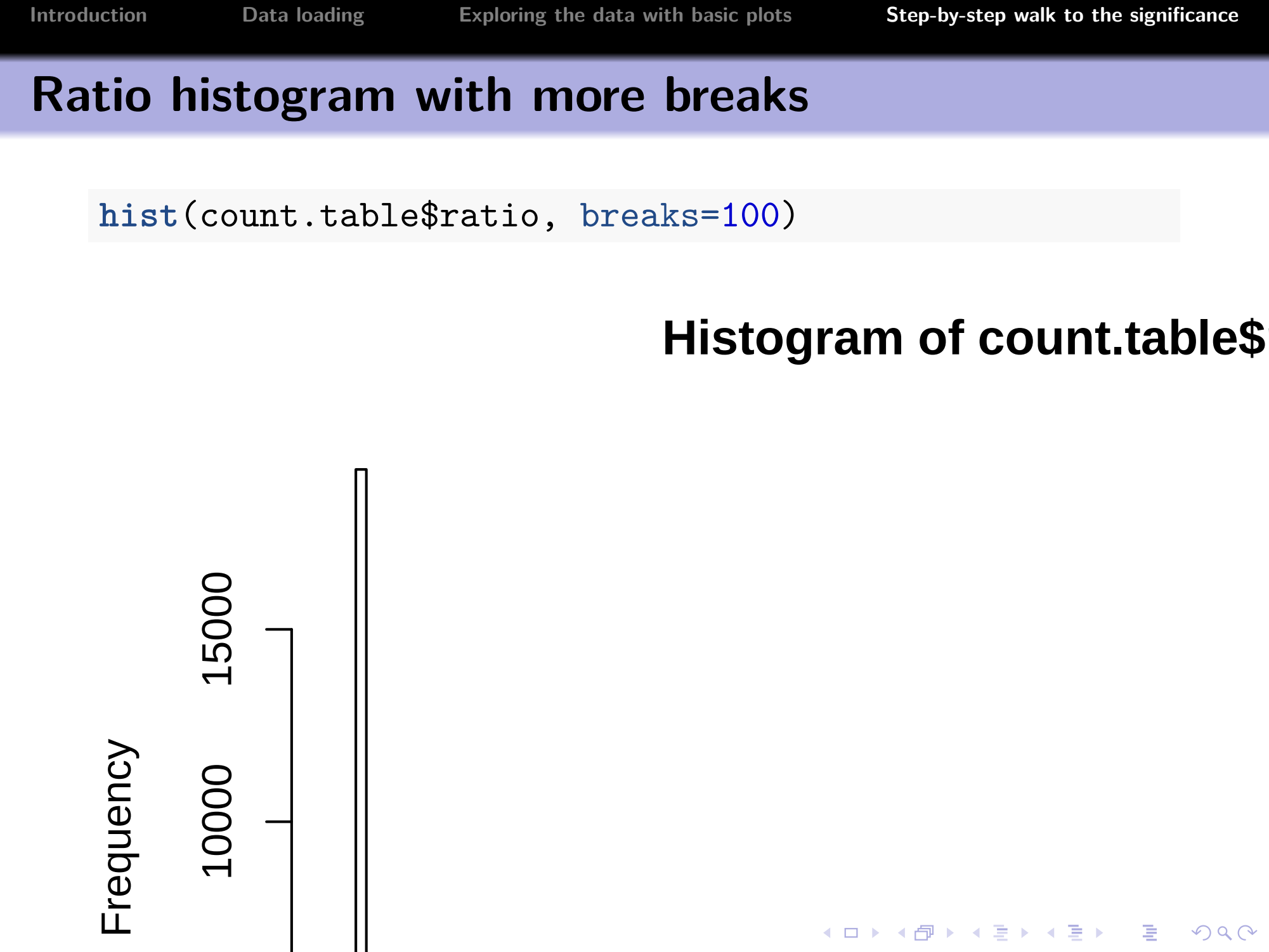
ChIP/input ratios

```
count.table$ratio <- (count.table$counts.chip + 1) / (count

hist(count.table$ratio)
```



Ratio histogram with more breaks

```
hist(count.table$ratio, breaks=100)
```



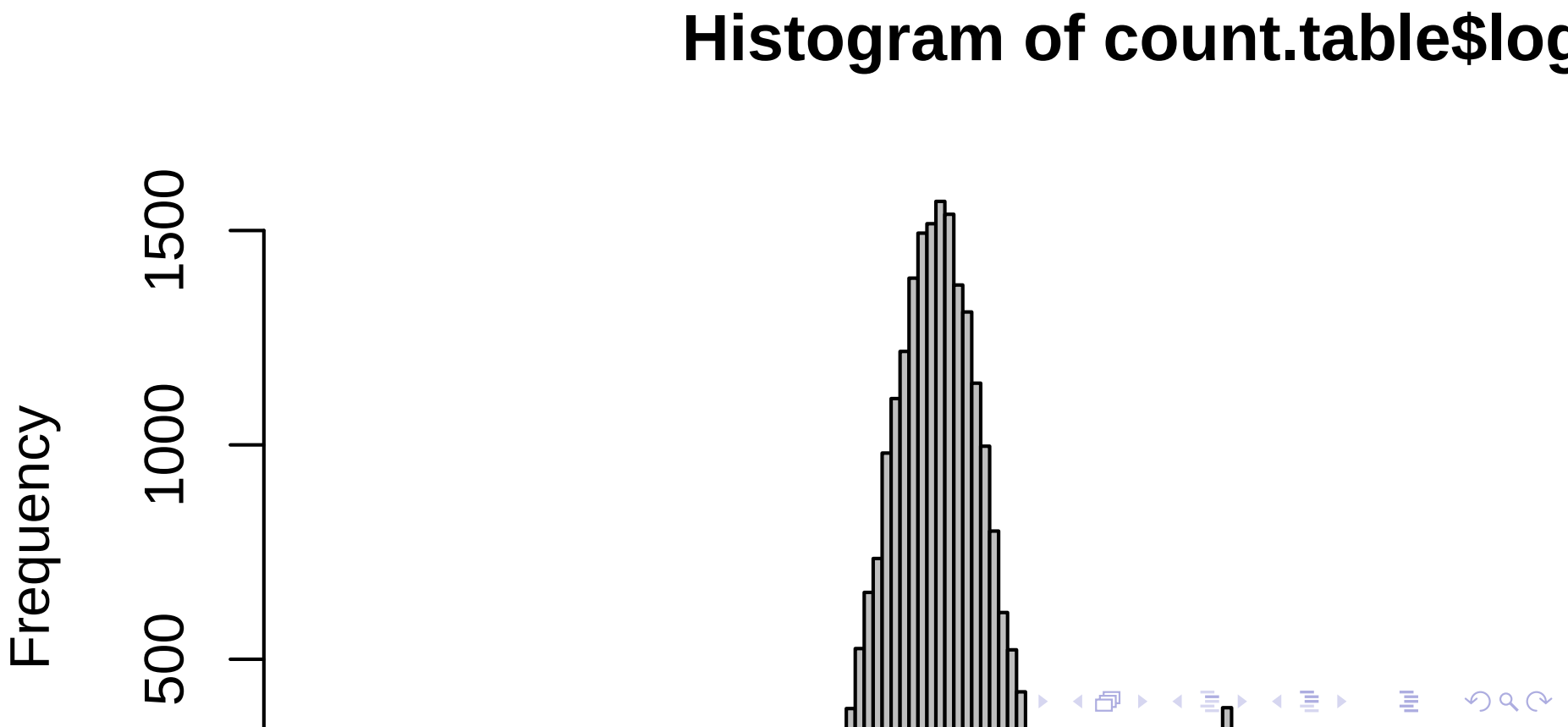
Ratio histogram with even more breaks

```
hist(count.table$ratio, breaks=1000)
```

Ratio histogram with truncated X axis

Let us truncate the X axis (and make it explicit on X axis label).

```
count.table$log2.ratio <- log2(count.table$ratio)
hist(count.table$log2.ratio, breaks = 200, xlim=c(-5,5), co
```



Checking library sizes

We visibly have a problem: almost all log2 ratios are $\ll 0$. Why ? Check library sizes.

```
sum(count.table$counts.chip)
```

```
## [1] 2622163
```

```
sum(count.table$counts.input)
```

```
## [1] 7480400
```

Count scaling

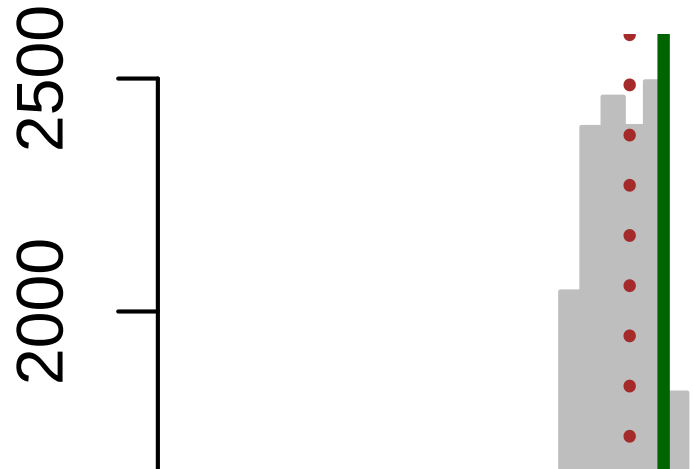
The simplest way to "normalize" is to scale input counts by library sum.

```
## Normalize input counts by library sizes
count.table$input.scaled.libsize <- count.table$counts.inpu

plot(x = count.table$input.scaled.libsize + 1,
     y = count.table$counts.chip + 1, col="darkviolet", log
     main="ChIP versus input",
     xlab="Scaled input counts",
     ylab="ChIP counts")
grid()
abline(a=0, b=1, col="black")
```

Log-ratios

```
count.table$scaled.ratio.libsum <- (count.table$counts.chip

hist(count.table$scaled.ratio.libsum, breaks=2000, xlim=c(
abline(v=mean(count.table$scaled.ratio.libsum), col="darkgr
abline(v=median(count.table$scaled.ratio.libsum), col="brow
legend("topright", c("mean", "median"), col=c("darkgreen",
```

Count ratio histogram



Print summary statistics

Let us compare the mean and median counts for ChIP and input samples.

```
summary(count.table[, c("counts.chip", "counts.input")])
```

```
##    counts.chip      counts.input
##  Min.   :    0   Min.   :  0.0
##  1st Qu.:   82   1st Qu.:277.0
##  Median :  101   Median :320.0
##  Mean   :  113   Mean   :322.4
##  3rd Qu.:  125   3rd Qu.:372.0
##  Max.   :12411   Max.   :691.0
```

- For the input, mean and medium are almost the same.
- For the ChIP, we have a 10% difference.
- This difference likely results from the "statistical outliers", i.e. our peaks.

Input normalization by median

Why? the median is very robust to outliers (to be discussed during the course).

```
## Normalize input counts by median count
count.table$input.scaled.median <- count.table$counts.input

count.table$scaled.ratio.median <- (count.table$counts.chip

## Print the mean and median scaled ratios
mean(count.table$scaled.ratio.median)
```

```
## [1] 1.101867
```

```
median(count.table$scaled.ratio.median)
```

```
## [1] 1.006283
```

Count ratio distribution after median scaling

```
hist(count.table$scaled.ratio.median, breaks=2000, xlim=c(0
abline(v=mean(count.table$scaled.ratio.median), col="darkgr
abline(v=median(count.table$scaled.ratio.median), col="brow
legend("topright", c("mean", "median"), col=c("darkgreen",
```

Count ratio histogram