csaw: ChIP-seq analysis with windows

User's Guide

Aaron Lun

First edition 15 August 2012

Last revised 1 July 2014

Contents

Chapter 1

Introduction

1.1 Scope

This document gives an overview of the Bioconductor package csaw for detecting differential binding (DB) in ChIP-seq experiments. Specifically, csaw uses sliding windows to identify significant changes in binding patterns for transcription factors (TFs) or histone marks across different biological conditions. However, it can also be applied to any sequencing technique where reads represent coverage of enriched genomic regions. The statistical methods described here are based upon those in the edgeR package [Robinson et al., 2010]. Knowledge of edgeR is useful but not a prerequesite for reading this guide.

1.2 How to get help

Most questions about csaw should be answered by the documentation. Every function mentioned in this guide has its own help page. For example, a detailed description of the arguments and output of the `windowCounts` function can be obtained by typing `?windowCounts` or `help(windowCounts)` at the R prompt. Further detail on the methods or the underlying theory can be found in the references at the bottom of each help page.

The authors of the package always appreciate receiving reports of bugs in the package functions or in the documentation. The same goes for well-considered suggestions for improvements. Other questions about how to use csaw are best sent to the Bioconductor mailing list `bioconductor@stat.math.ethz.ch`. To subscribe to the mailing list, see `https://stat.ethz.ch/mailman/listinfo/bioconductor`. Please send requests for general assistance and advice to the mailing list rather than to the individual authors.

Users posting to the mailing list for the first time may find it helpful to read the posting guide at `http://www.bioconductor.org/doc/postingGuide.html`.

1.3 Quick start

A typical ChIP-seq analysis would look something like the code described below. This assumes that a vector of file paths to sorted and indexed BAM files is provided in `bam.files` and a design matrix in supplied in `design`.

```
> require(csaw)
> data <- windowCounts(bam.files, ext=110)
> binned <- windowCounts(bam.files, bin=TRUE, width=10000)
> normfacs <- normalizeChIP(binned$counts, lib.size=binned$totals)
> require(edgeR)
> y <- DGEList(data$counts, lib.size=data$totals, norm.factors=normfacs)
> y <- estimateDisp(y, design)
> results <- glmQLFTest(y, design, robust=TRUE)
> merged <- mergeWindows(data$region, tol=1000L)
> tabcom <- combineTests(merged$id, results$table)
```

For anyone still reading, the csaw analysis pipeline can be summarized into several steps:

1. Loading in data from BAM files.
2. Calculating normalization factors.
3. Filtering out uninteresting regions.
4. Identifying DB windows.
5. Correcting for multiple testing.

Each step will be demonstrated in this guide with some publicly available data. The dataset below focuses on changes in NFYA binding between embryonic stem cells and terminal neurons [Tiwari et al., 2012]. This will be used as a case study for most of the code examples throughout the guide.

```
> bam.files <- c("es_1.bam", "es_2.bam", "tn_1.bam", "tn_2.bam")
> design <- model.matrix(~factor(c('es', 'es', 'tn', 'tn')))
> colnames(design) <- c("intercept", "cell.type")
```

A comprehensive listing of the datasets used in this guide is provided in Section 8.1, along with instructions on how to obtain and process them for entry into the csaw pipeline.

Chapter 2

Converting reads to counts

Hello, reader. A little box like this will be present at the start of each chapter. It's intended to tell you which objects from previous chapters are needed to get the code in the current chapter to work. At this point, all we need are the `bam.files` that we defined in the introduction above.

2.1 Types of input data

Sorted and indexed BAM (i.e. binary SAM) files are required as input into the read counting functions in csaw. Sorting should be performed on read position. Each index file should be named as `xxx.bam.bai` for the BAM file named `xxx.bam`. Both files should also be in the same directory. Users should be aware that the sensibility of the supplied index is not checked prior to counting. A common mistake is to replace or update the BAM file without updating the index. This will cause csaw to return incorrect results when it attempts to load alignments from new BAM file.

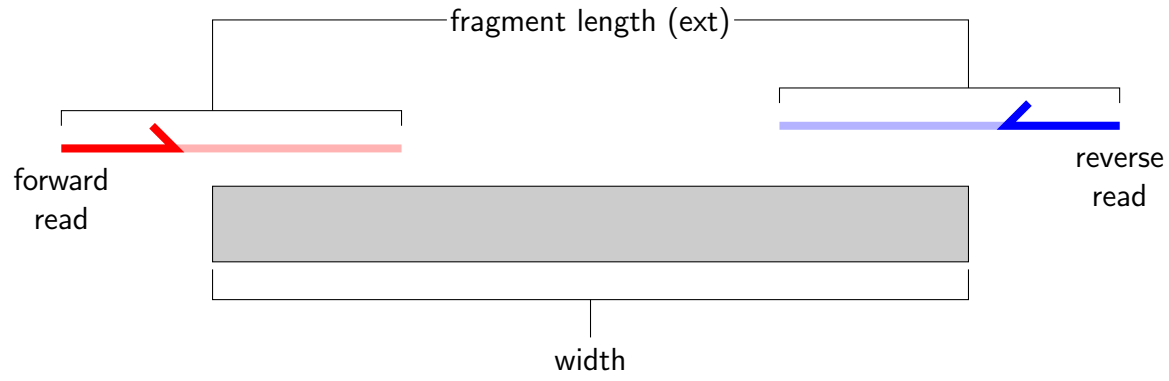
2.2 Counting reads into windows

2.2.1 Overview

The `windowCounts` function uses a sliding window approach to count fragments for a set of libraries. For single-end data, the fragment corresponding to a read is imputed by directionally extending each read to the average fragment length. The number of fragments overlapping a genomic window is counted. This is repeated after sliding the window along the genome to a new position. A count is then obtained for each window in each library.

```
> frag.len <- 110
> data <- windowCounts(bam.files, ext=frag.len, width=10)
> head(data$counts)

     [,1] [,2] [,3] [,4]
[1,]    6    7    4    4
[2,]    1    2    9   13
[3,]    0    2    9   13
[4,]   13    5    4    1
[5,]   16    5    5    1
[6,]   10   13   21    5

> head(data$region)

GRanges with 6 ranges and 0 metadata columns:
      seqnames             ranges strand
         <Rle>          <IRanges>  <Rle>
  [1]     chr1 [3003751, 3003760]      *
  [2]     chr1 [3012951, 3012960]      *
  [3]     chr1 [3013001, 3013010]      *
  [4]     chr1 [3043151, 3043160]      *
  [5]     chr1 [3043201, 3043210]      *
  [6]     chr1 [3065501, 3065510]      *
  ---
  seqlengths:
                   chr1                chr10 ... chrY_JH584303_random
              195471971            130694993 ...                158099
```

For single-end data, suitable values for the average fragment length in `ext` can be estimated from the primary peak in a cross-correlation plot (see Section 2.4). Alternatively, the length can be estimated from diagnostics during ChIP or library preparation, e.g., post-fragmentation gel electrophoresis images. Typical values range from 100 to 300 bp, depending on the efficiency of sonication and the use of size selection steps in library preparation.



The specified `width` of each window controls the compromise between spatial resolution and count size. Larger windows will yield higher read counts which can provide more power for DB detection. However, spatial resolution is also lost for large windows whereby adjacent features can no longer be distinguished. Reads from a DB site may be counted alongside reads from a non-DB site (e.g., non-specific background) or even those from an adjacent site that is DB in the opposite direction. This will result in the loss of DB detection power.

The window size can be interpreted as a measure of the width of the binding site. Thus, TF analyses will usually use a window size of several base pairs. This maximizes spatial resolution which is critical for narrow regions of enrichment. For histone marks, widths of at least 150 bp are recommended [Humburg et al., 2011]. This corresponds to the length of DNA wrapped up in each nucleosome, i.e., the smallest relevant unit for histone mark enrichment. For diffuse marks, the sizes of enriched regions are more variable and so the compromise between resolution and power is more arbitrary. Analyses with multiple widths can be combined to provide a comprehensive picture of DB at all resolutions.

2.2.2 Filtering out low-quality reads

Reads that have been marked as PCR duplicates in the SAM flag can be ignored by setting `dedup=TRUE`. This can reduce the variability caused by inconsistent duplication between replicates. However, it also caps the number of reads at each position. This can lead to loss of DB detection power in high abundance regions. Spurious differences may also be introduced when the same upper bound is applied to libraries of varying size. Thus, duplicate removal is not recommended for routine DB analyses. Of course, removal may be unavoidable in some cases, e.g., involving libraries generated from low quantities of DNA.

Reads can also be filtered out based on the minimum mapping score with the `minq` argument. Low mapping scores are indicative of incorrectly and/or non-uniquely aligned sequences. Removal of these reads is highly recommended as it will ensure that only the reliable alignments are supplied to csaw. The exact value of the threshold depends on the range of scores provided by the aligner. The subread program [Liao et al., 2013] was used to align the reads in this dataset so a value of 100 might be appropriate.

```
> demo <- windowCounts(bam.files, ext=frag.len, minq=100, dedup=TRUE)
```

On a more subjective note, reads on particular chromosomes can be specifically counted by specifying the chromosomes of interest in `restrict`. This avoids the need to count reads on unassigned contigs or uninteresting chromosomes, e.g., the mitochondrial genome for ChIP-seq studies targeting nuclear factors. Alternatively, it allows `windowCounts` to work on huge datasets or in limited memory by analyzing only one chromosome at a time.

Finally, reads lying in certain regions can also be removed by specifying those regions in `discard`. This is intended to remove reads that are wholly aligned within known repeat regions but were not removed by the `minq` filter. Repeats are problematic as changes in repeat copy number between conditions can lead to spurious DB. Removal of reads within

repeat regions can avoid detection of these irrelevant differences. Annotated repeats can be found for a number of species on the UCSC website, e.g., mouse.

```
> repeats <- GRanges("chr1", IRanges(3000001, 3002128))
> demo <- windowCounts(bam.files, ext=frag.len, discard=repeats,
+     restrict=c("chr1", "chr10", "chrX"))
```

Using `discard` is safer than simply ignoring windows that overlap the repeats. For example, a large window might contain both repeat regions and non-repeat regions. Discarding the window because of the former will compromise detection of DB features in the latter. Of course, any DB sites within the discarded regions will obviously be lost from downstream analyses. Some caution is therefore required when specifying the regions of disinterest.

2.2.3 Increasing speed and memory efficiency

The `spacing` parameter controls the distance with which windows are shifted to the next position in the genome. Using a higher value will reduce the computational load as fewer counts are extracted for downstream analysis. This may be useful when machine memory is limited. Of course, this also sacrifices spatial resolution as adjacent positions are not counted and thus cannot be distinguished.

```
> demo <- windowCounts(bam.files, spacing=100, ext=frag.len)
> head(demo$region)

GRanges with 6 ranges and 0 metadata columns:
      seqnames             ranges strand
         <Rle>          <IRanges>  <Rle>
  [1]     chr1 [3013001, 3013001]      *
  [2]     chr1 [3043201, 3043201]      *
  [3]     chr1 [3065501, 3065501]      *
  [4]     chr1 [3089401, 3089401]      *
  [5]     chr1 [3241701, 3241701]      *
  [6]     chr1 [3254601, 3254601]      *
  ---
  seqlengths:
                   chr1                chr10 ... chrY_JH584303_random
              195471971            130694993 ...                158099
```

For analyses with large windows, it is also worth increasing the `spacing` to a fraction of the specified `width`. This reduces the computational work by decreasing the number of windows and extracted counts. Any loss in spatial resolution due to a larger spacing interval is negligble when compared to that already lost by using a large window size.

Windows with a low sum of counts across all libraries can be filtered out using the `filter` argument. This improves memory efficiency by discarding the majority of low-abundance windows corresponding to uninteresting background regions. The default filter value is set as

the number of libraries multiplied by 5. This roughly corresponds to the minimum average count required for accurate statistical modelling. Note that more sophisticated filtering is recommended and should be applied later (see Chapter 4).

```
> demo <- windowCounts(bam.files, ext=frag.len, filter=30)
> head(demo$counts)
```

```
     [,1] [,2] [,3] [,4]
[1,]   10   13   21    4
[2,]   10   12   18    4
[3,]    9    8   10    3
[4,]   22   18   21   12
[5,]   42   38   40   29
[6,]   22   23   20   17
```

A special case occurs when `bin=TRUE`. This will set `spacing=width` and only use the 5′ end of each read for counting. Reads are then counted into contiguous bins across the genome. No filtering is performed such that a count value must be returned for each bin. Users should set `width` to a reasonably large value, otherwise reads will be counted and reported for every single base in the genome by default.

```
> demo <- windowCounts(bam.files, width=1000, bin=TRUE)
> head(demo$region)
```

```
GRanges with 6 ranges and 0 metadata columns:
      seqnames             ranges strand
         <Rle>          <IRanges>  <Rle>
  [1]     chr1 [3000001, 3001000]      *
  [2]     chr1 [3001001, 3002000]      *
  [3]     chr1 [3002001, 3003000]      *
  [4]     chr1 [3003001, 3004000]      *
  [5]     chr1 [3004001, 3005000]      *
  [6]     chr1 [3005001, 3006000]      *
  ---
  seqlengths:
                   chr1                chr10 ... chrY_JH584303_random
              195471971            130694993 ...                158099
```

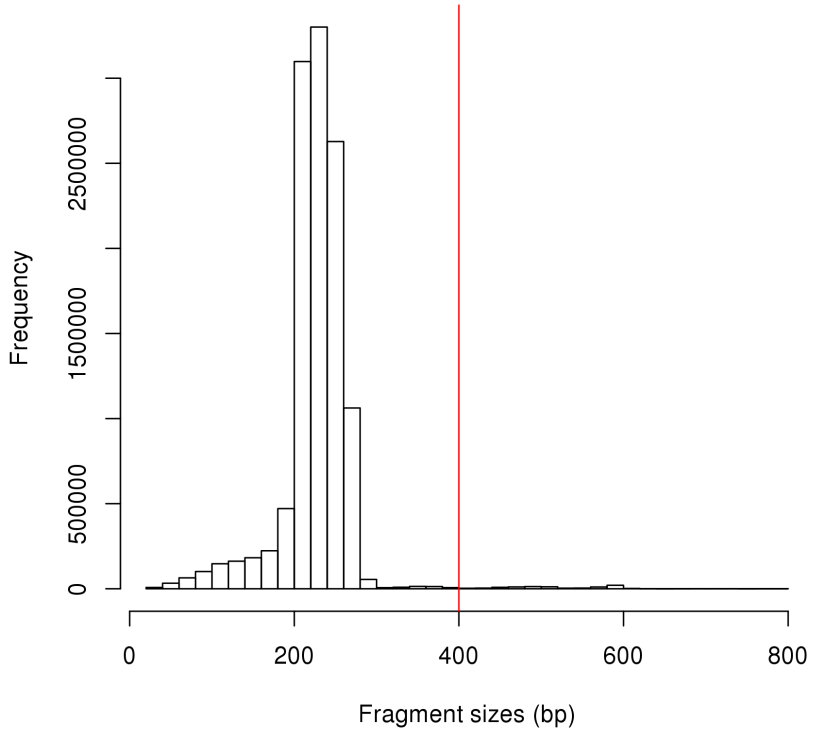
2.3 Experiments involving paired-end tags

ChIP experiments with paired-end sequencing can be accomodated by setting `pet="both"` in the `windowCounts` function. Read extension is not required for paired-end data as the genomic interval spanned by the originating fragment is explicitly defined between the 5′ positions of the paired reads. The number of fragments overlapping each window is then counted as described. By default, only proper pairs are used whereby the two reads are on the same chromosome, face inward and are no more than `max.frag` apart.

```
> petBamFile <- "example-pet.bam"
> demo <- windowCounts(petBamFile, max.frag=400, pet="both")
```

A suitable value for `max.frag` can be chosen by examining the distribution of fragment sizes using the `getPETSizes` function. In this example, a user might use a value of around 500 bp as it covers most of the fragment size distribution. The plot can also be used to examine the quality of the PET sequencing procedure. The location of the peak should be consistent with the fragmentation and size selection steps in library preparation.

```
> out <- getPETSizes(petBamFile)
> frag.sizes <- out$sizes[out$sizes<=800]
> hist(frag.sizes, breaks=50, xlab="Fragment sizes (bp)", ylab="Frequency", main="")
> abline(v=400, col="red")
```



The number of fragments exceeding the maximum size can be recorded for quality control. The `getPETSizes` function also returns the number of invalid pairs, inter-chromosomal pairs, pairs with one unmapped read and single reads. A non-negligble proportion of these reads indicates that there may be some problems with the paired-end alignment or sequencing.

```
> c(out$diagnostics, too.large=sum(out$sizes > 400))

        total        single    unoriented mate.unmapped     inter.chr
     30854223             0        254594        888175       2928351
    too.large
       215607
```

In cases where there are a non-negligble proportion of invalid pairs, the reads can be rescued by setting `rescue.pairs=TRUE`. For each invalid intra-chromosomal read pair, the read with the higher mapping quality score will be directionally extended to `ext` to impute the fragment. For inter-chromosomal read pairs, both reads are extended in this manner. Counting will then be performed with these fragments in addition to those from the valid pairs. A value of `ext` can be chosen based on the mode of the distribution above.

```
> demo <- windowCounts(petBamFile, max.frag=400, pet="both", ext=200, rescue.pairs=TRUE)
```

Paired-end data can also be treated as single-end data by specifiying `pet="first"` or `pet="second"`. This will only take the first or second read of each pair as defined in the SAM flags. Unlike `rescue.pairs`, the selection and use of one read is done for all read pairs regardless of validity. This may be useful for comparing paired-end with single-end data, or in truly disastrous situations where paired-end sequencing has failed.

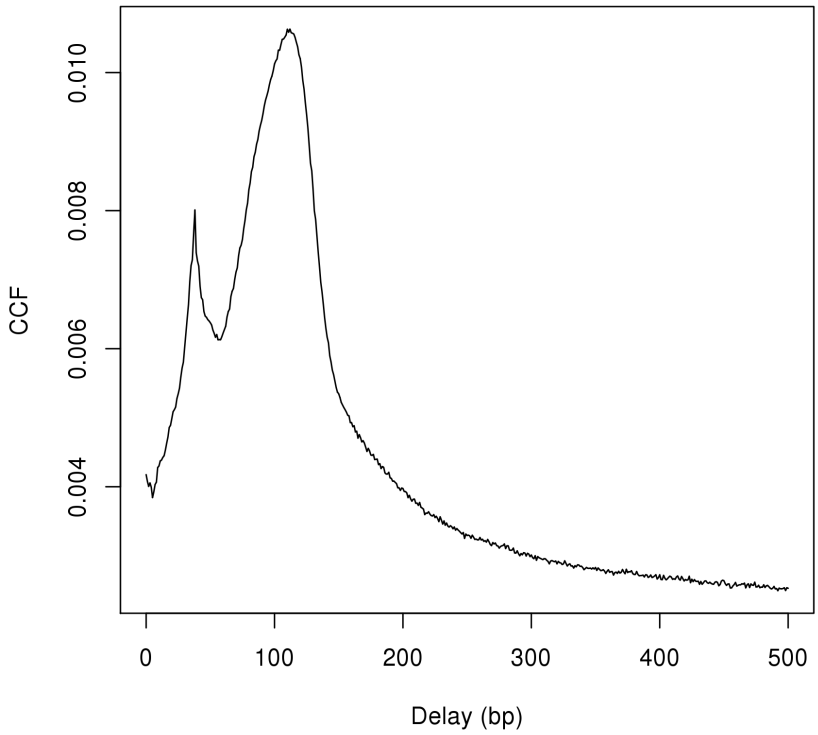
```
> demo <- windowCounts(petBamFile, max.frag=400, pet="first")
```

Note that all of the paired-end methods in csaw depend on the synchronisation of mate information for each alignment in the BAM file. Any file manipulations that might break this synchronisation should be corrected prior to the use of csaw.

2.4 Estimating the average fragment length

Cross-correlation plots can be generated directly from BAM files using the `correlateReads` function. This provides a measure of the immunoprecipitation (IP) efficiency of a ChIP-seq experiment [Kharchenko et al., 2008]. Efficient IP should yield a smooth peak at a distance corresponding to the average fragment length. This reflects the strand-dependent bimodality of reads around narrow regions of enrichment, e.g., TF binding sites. The location of the peak can be used as an estimate of the average fragment length ($\sim$110 bp below) for read extension in `windowCounts`.

```
> max.delay <- 500
> x <- correlateReads(bam.files, max.delay, dedup=TRUE, cross=TRUE, minq=100)
> plot(0:max.delay, x, type="l", ylab="CCF", xlab="Delay (bp)")
```

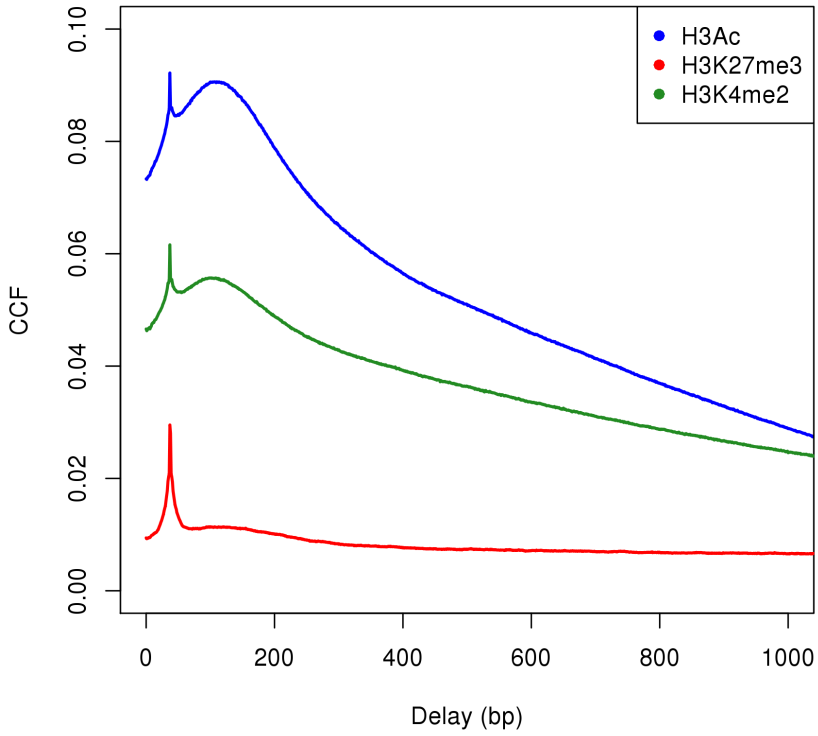


A sharp spike may also observed in the plot at a distance corresponding to the read length. This is thought to be an artifact, caused by the preference of aligners towards uniquely mapped reads. The size of the smooth peak can be compared to the height of the spike to assess the signal-to-noise ratio of the data [Landt et al., 2012]. Poor IP efficiency will result in a smaller or absent peak as bimodality is less pronounced. Note that duplicate removal is required here to reduce the size of the read length spike. Otherwise, the fragment length peak will not be visible as a separate entity.

Cross-correlation plots can also be used for fragment length estimation of narrow histone marks such as histone acetylation and H3K4 methylation. However, they are less effective for regions of diffuse enrichment where bimodality is not obvious (e.g. H3K27 trimethylation).

```
> n <- 10000
> h3ac <- correlateReads("h3ac.bam", n, dedup=TRUE, cross=TRUE)
> h3k27me3 <- correlateReads("h3k27me3.bam", n, dedup=TRUE, cross=TRUE)
> h3k4me2 <- correlateReads("h3k4me2.bam", n, dedup=TRUE, cross=TRUE)
> plot(0:n, h3ac, col="blue", ylim=c(0, 0.1), xlim=c(0, 1000),
+     xlab="Delay (bp)", ylab="CCF", pch=16, type="l", lwd=2)
> lines(0:n, h3k27me3, col="red", pch=16, lwd=2)
> lines(0:n, h3k4me2, col="forestgreen", pch=16, lwd=2)
```

```
> legend("topright", col=c("blue", "red", "forestgreen"),
+     c("H3Ac", "H3K27me3", "H3K4me2"), pch=16)
```



2.5 Ensuring synchronisation

In practice, most ChIP-seq analyses using csaw will involve multiple calls to the `windowCounts` function. Users should supply the same values for the `bam.files`, `dedup`, `pet`, `minq`, `restrict`, `discard` and (for paired-end data) `max.frag` and `rescue.pairs` parameters at each call. This ensures that the same reads are being used for counting throughout the analysis. For complex analyses, this synchronisation can be easily maintained by constructing a parameter list and calling the `countWindows` wrapper function.

```
> param <- list(bam.files=bam.files, dedup=TRUE, minq=100, pet="none", ext=frag.len)
> demo <- countWindows(param, filter=50)
> demo <- countWindows(param, bin=TRUE, width=1000)
```

This strategy avoids the need for repeated manual specification of non-default arguments at each function call. Global changes can then be implemented by altering the contents of

param directly. A good check for synchronisation is to ensure that the values of ...$totals are identical between calls. This means that the same reads are extracted from the BAM file in each call. For simplicity, the examples in the rest of this guide will use the windowCounts function. This performs read counting with the default values for most of the critical parameters except for bam.files, ext and width (see the call to generate data, above).

2.6 Counting over manually specified regions

The csaw package focuses on counting reads into windows. However, it may be desirable on occasion to use the same conventions (e.g., duplicate removal, quality score filtering) when counting reads into pre-specified regions. This can be performed with the regionCounts function, which is largely a wrapper for findOverlaps from the GenomicRanges package.

```
> my.regions <- GRanges(c("chr11", "chr12", "chr15"),
+     IRanges(c(75461351, 95943801, 21656501),
+     c(75461610, 95944810, 21657610)))
> reg.counts <- regionCounts(bam.files, my.regions, ext=frag.len, minq=100, dedup=TRUE)
> reg.counts$counts

     [,1] [,2] [,3] [,4]
[1,]   35   53   88   81
[2,]    6    5    7    9
[3,]   13   11    8   15
```

Chapter 3

Calculating normalization factors

This next chapter will need the `bam.files` vector again. You'll notice that that a number of other BAM files are used in this chapter. However, these are just for demonstration purposes and aren't necessary for the main example.

3.1 Eliminating composition biases

3.1.1 Using the TMM method on binned counts

As the name suggests, composition biases are formed when there are differences in the composition of sequences across libraries. Highly enriched regions consume more sequencing resources and thereby suppress the representation of other regions. Differences in the magnitude of suppression can lead to spurious DB calls. Scaling by library size fails to correct for this as composition biases can still occur in libraries of the same size.

To remove composition biases in csaw, reads are counted in large bins and the counts are used for normalization with the `normalizeChIP` wrapper function. This uses the trimmed mean of M-values (TMM) method [Robinson and Oshlack, 2010] to correct for any sytematic fold change in the coverage of the bins. The assumption here is that most bins represent non-DB background regions so any consistent difference across bins must be spurious.

```
> binned <- windowCounts(bam.files, bin=TRUE, width=10000)
> normfacs <- normalizeChIP(binned$counts, lib.size=binned$totals)
> normfacs
```

```
[1] 0.9843100 0.9594416 1.0334848 1.0245790
```

The TMM method trims away putative DB bins (i.e., those with extreme M-values) and computes normalization factors from the remainder to use in edgeR. The size of each library

is scaled by the corresponding factor to obtain an effective library size for modelling. A larger normalization factor results in a larger effective library size and is conceptually equivalent to scaling each individual count downwards, given that the ratio of that count to the library size will be smaller. Check out the edgeR user's guide for more information.

Note that `normalizeChIP` skips the precision weighting step in the TMM method by default. Weighting increases the contribution of bins with high counts. However, these bins are more likely to contain binding sites and thus are more likely to be DB. If any DB regions should survive trimming (e.g., those with less extreme fold changes), upweighting them would be counterproductive. In fact, users may wish to explicitly filter out such bins and run TMM only on putative background regions, as shown below.

```
> ab <- aveLogCPM(binned$counts, lib.size=binned$total)
> keep <- ab <= quantile(ab, p=0.9)
> normalizeChIP(binned$counts[keep,], lib.size=binned$total)

[1] 0.9859577 0.9436046 1.0432673 1.0302819
```

3.1.2 Choosing a bin size

By definition, read coverage is low for background regions. This can lead to a high frequency of zero counts and undefined M-values when reads are counted into windows. Adding a prior count is only a superficial solution as the chosen prior will have undue influence on the estimate of the normalization factor when many counts are low. The variance of the fold change distribution is also higher for low counts. This reduces the effectiveness of the trimming procedure during normalization. These problems can be overcome by using large bins to increase the size of the counts prior to TMM normalization.

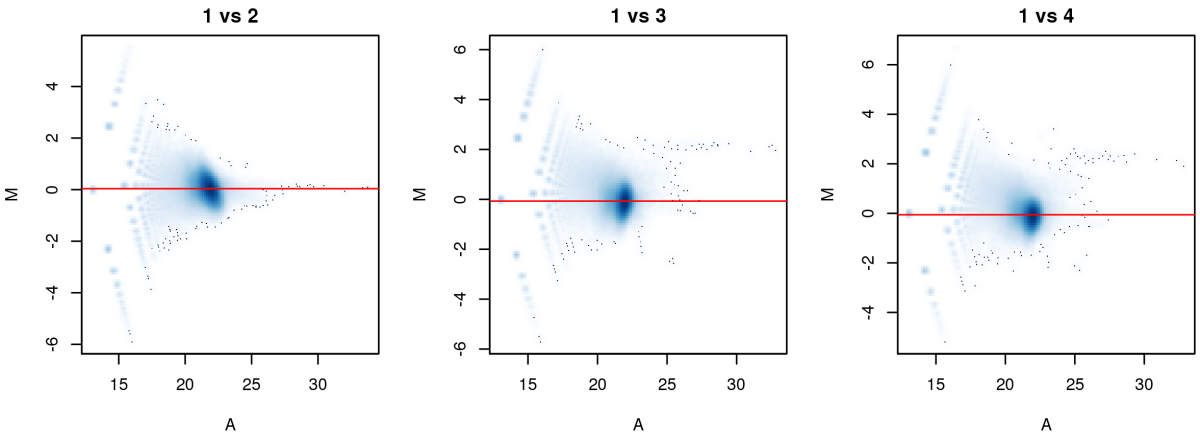
Of course, this strategy requires the user to supply a bin size. Excessively large bins are problematic as background and enriched regions will be included in the same bin. This makes it difficult to trim away enriched bins during the TMM procedure. Obviously, bins which are too small will have read counts which are too low. Testing multiple bin sizes is recommended to ensure that the estimates are robust to any changes. Experience suggests that a value around 10000 bp is suitable for most datasets.

```
> demo <- windowCounts(bam.files, bin=TRUE, width=5000)
> normalizeChIP(demo$counts, lib.size=demo$total)

[1] 0.9840325 0.9611444 1.0318386 1.0246845

> demo <- windowCounts(bam.files, bin=TRUE, width=15000)
> normalizeChIP(demo$counts, lib.size=demo$total)

[1] 0.9848029 0.9578794 1.0344662 1.0247632
```

3.1.3 Visualizing normalization efforts with an MA plot

The effectiveness of normalization can be checked using a MA plot. A single main cloud of points should be present that represents the background regions. Separation into multiple discrete points indicates that the counts are too low and that larger bin sizes should be used. Composition biases manifest as a vertical shift in the position of this cloud. Ideally, the log-ratios of the corresponding normalization factors should correspond to the centre of the cloud. This indicates that undersampling has been identified and corrected.

```
> par(mfrow=c(1, 3), mar=c(5, 4, 2, 1.5))
> adj.counts <- cpm(binned$counts, log=TRUE)
> for (i in 1:(length(bam.files)-1)) {
+     cur.x <- adj.counts[,1]
+     cur.y <- adj.counts[,1+i]
+     smoothScatter(x=(cur.x+cur.y)/2+6*log2(10), y=cur.x-cur.y,
+         xlab="A", ylab="M", main=paste("1 vs", i+1))
+     all.dist <- diff(log2(normfacs[c(i+1, 1)]))
+     abline(h=all.dist, col="red")
+ }
```



3.2 Eliminating efficiency biases

Efficiency biases are commonly observed in ChIP-seq data. This refers to fold changes in enrichment that are introduced by variability in IP efficiencies between libraries. These technical differences are of no biological interest and must be removed. This can be achieved by assuming some top percentage of bins or windows with the highest abundances contain binding sites. The TMM method can then be applied to eliminate systematic differences in the counts across those bins. In the example below, the top 1% of bins are assumed to contain binding sites. For consistency, the bin size in Section 3.1.1 is re-used here though the counts for binding sites should be high enough for smaller bins.

```
> me.demo <- windowCounts(c("h3k4me3_mat.bam", "h3k4me3_pro.bam"), bin=TRUE, width=10000L)
> ab <- aveLogCPM(me.demo$counts, lib.sizes=me.demo$total)
> keep <- rank(ab) > 0.99*length(ab)
> me.norm <- normalizeChIP(me.demo$count[keep,], lib.sizes=me.demo$totals)
> me.norm
```

```
[1] 0.7108908 1.4066859
```

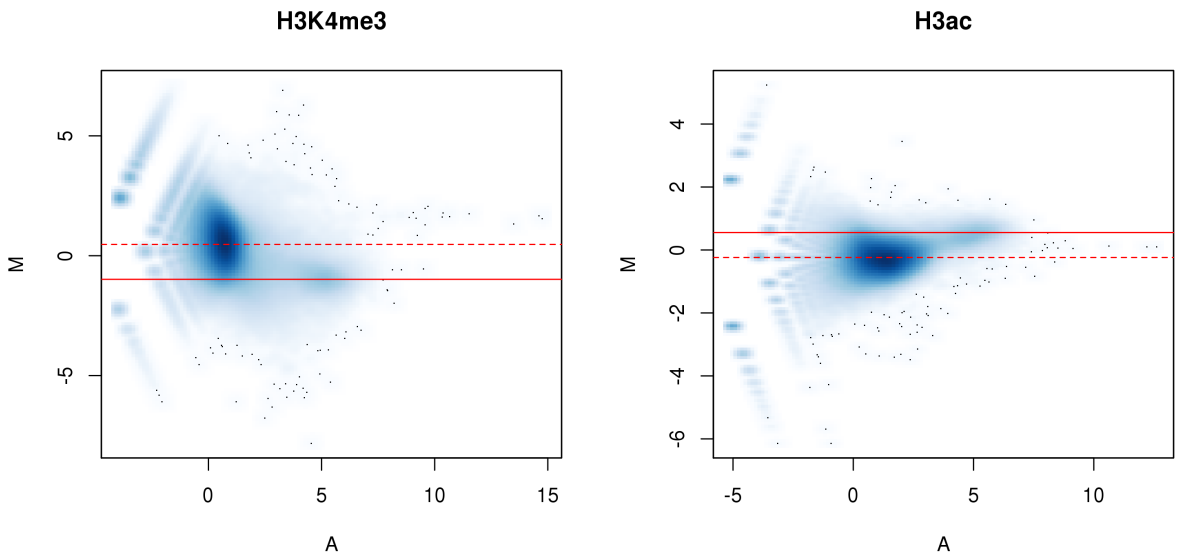
```
> ac.demo <- windowCounts(c("h3ac.bam", "h3ac_2.bam"), bin=TRUE, width=10000L)
> ab <- aveLogCPM(ac.demo$counts, lib.sizes=ac.demo$total)
> keep <- rank(ab) > 0.99*length(ab)
> ac.norm <- normalizeChIP(ac.demo$count[keep,], lib.sizes=ac.demo$totals)
> ac.norm
```

```
[1] 1.2114998 0.8254232
```

This method assumes that most high-abundance bins are not DB. Any systematic changes must be caused by differences in IP efficiency or some other technical issue. Genuine biological differences may be removed when the assumption of a non-DB majority does not hold, i.e., overall binding is truly lower in one condition. Also, the percentage of bins to use for normalization may not be obvious if there is no obvious demarcation on a MA plot.

Speaking of which, the results of normalization can again be visualized with MA plots. Of particular interest is the cloud of points on the plot at high A-values. This represents a systematic fold change in bound regions, either due to genuine DB or variable IP efficiency. Note the difference in the normalization factors from removal of efficiency bias (full) against that of composition bias (dashed). The choice between the two methods depends on whether one assumes that the systematic differences at high abundances represent genuine DB events. If not, they must represent efficiency biases and should be removed.

```
> par(mfrow=c(1,2))
> for (it in 1:2) {
+     if (it==1) {
+         demo <- me.demo
+         norm <- me.norm
+         main <- "H3K4me3"
+     } else {
+         demo <- ac.demo
+         norm <- ac.norm
+         main <- "H3ac"
+          }
+         adjc <- cpm(demo$counts, log=TRUE)
+         smoothScatter(x=rowMeans(adjc), y=adjc[,1]-adjc[,2],
+         xlab="A", ylab="M", main=main)
+     abline(h=log2(norm[1]/norm[2]), col="red")
+     compo.fac <- normalizeChIP(demo$counts, lib.sizes=demo$totals)
+     abline(h=log2(compo.fac[1]/compo.fac[2]), col="red", lty=2)
+ }
```



3.3 Dealing with trended biases

In more extreme cases, the bias may vary with the average abundance to form a trend. One possible explanation is that changes in IP efficiency will have little effect at low-abundance background regions and more effect at high-abundance binding sites. Thus, the magnitude of the bias between libraries will change with abundance. The trend cannot be corrected with scaling methods as no single scaling factor will remove differences at all abundances. Rather, non-linear methods are required such as cyclic loess or quantile normalization.

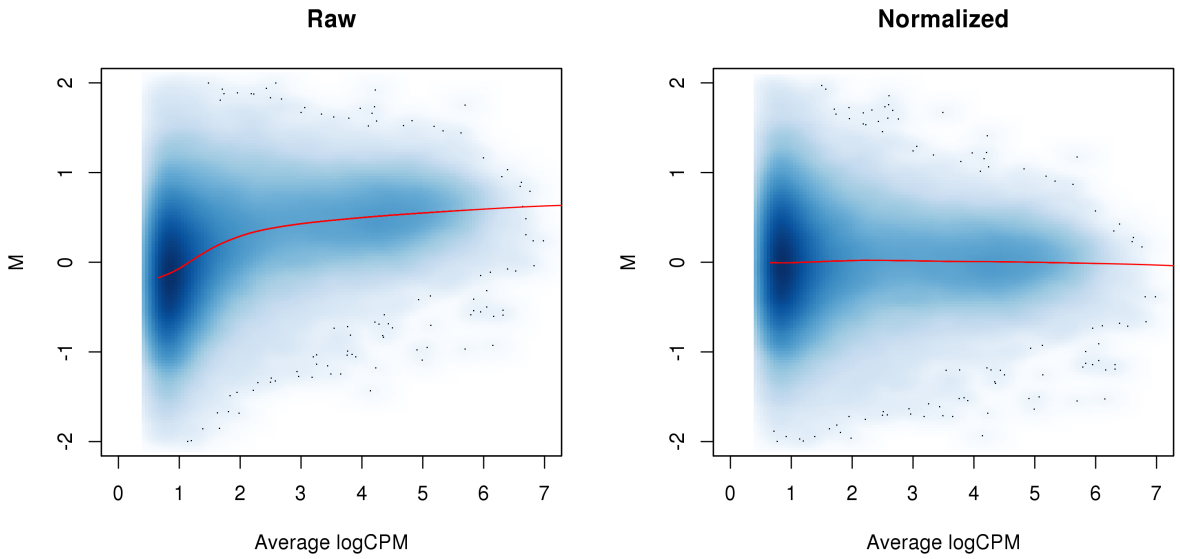
One such implementation is provided in `normalizeChIP` by setting `type="loess"`. This is based on the fast loess algorithm [Ballman et al., 2004] with minor adaptations to count data. An offset matrix is produced that contains the log-effective library sizes for each supplied bin/window. This can be used in edgeR, assuming that said bins or windows are also the ones to be tested for DB. The example below performs non-linear normalization on the counts for smaller bins, of which the top 5% are assumed to contain binding sites. Some filtering is necessary to remove low-abundance regions where loess curve fitting is inaccurate.

```
> ac.demo2 <- windowCounts(c("h3ac.bam", "h3ac_2.bam"), bin=TRUE, width=2000L)
> ab <- aveLogCPM(ac.demo2$counts, lib.size=ac.demo2$total)
> keep <- rank(ab) > 0.95*length(ab)
> ac.off <- normalizeChIP(ac.demo2$counts[keep,], lib.size=ac.demo2$totals, type="loess")
> head(ac.off)

           [,1]        [,2]
[1,] 0.05424593 -0.05424593
[2,] 0.03651008 -0.03651008
[3,] 0.04007449 -0.04007449
```

```
[4,] 0.14849385 -0.14849385
[5,] 0.08960421 -0.08960421
[6,] 0.25959074 -0.25959074
```

MA plots can be examined to determine whether normalization was successful. Any abundance-dependent trend in the M-values should be eliminated. If filtering needs to be performed prior to normalization, the average count computed by `aveLogCPM` is strongly recommended as the filter statistic (see Chapter 4 for more detail on choices of filter value). This is because an average count threshold will act as a clean vertical cutoff in the plots below. Thus, spurious trends that might affect normalization are not introduced.

```
> par(mfrow=c(1,2))
> aval <- ab[keep]
> o <- order(aval)
> adjc <- cpm(ac.demo2$counts[keep,], log=TRUE, lib.size=ac.demo2$total)
> mval <- adjc[,1]-adjc[,2]
> fit <- loessFit(x=aval, y=mval)
> smoothScatter(aval, mval, ylab="M", xlab="Average logCPM",
+     main="Raw", ylim=c(-2,2), xlim=c(0, 7))
> lines(aval[o], fit$fitted[o], col="red")
> #
> # Repeating after normalization.
> re.adjc <- log2(ac.demo2$counts[keep,]+0.5) - ac.off/log(2)
> mval <- re.adjc[,1]-re.adjc[,2]
> fit <- loessFit(x=aval, y=mval)
> smoothScatter(ab[keep], re.adjc[,1]-re.adjc[,2], ylab="M",
+     xlab="Average logCPM", main="Normalized", ylim=c(-2,2), xlim=c(0, 7))
> lines(aval[o], fit$fitted[o], col="red")
```



Note that all non-linear methods assume that most bins/windows are not DB at each abundance. This tends to be a stronger assumption than that for scaling methods, which only require a non-DB majority across all bins. Removal of the trend may not be appropriate if it represents some genuine biological phenomenon.

3.4 A word on other biases

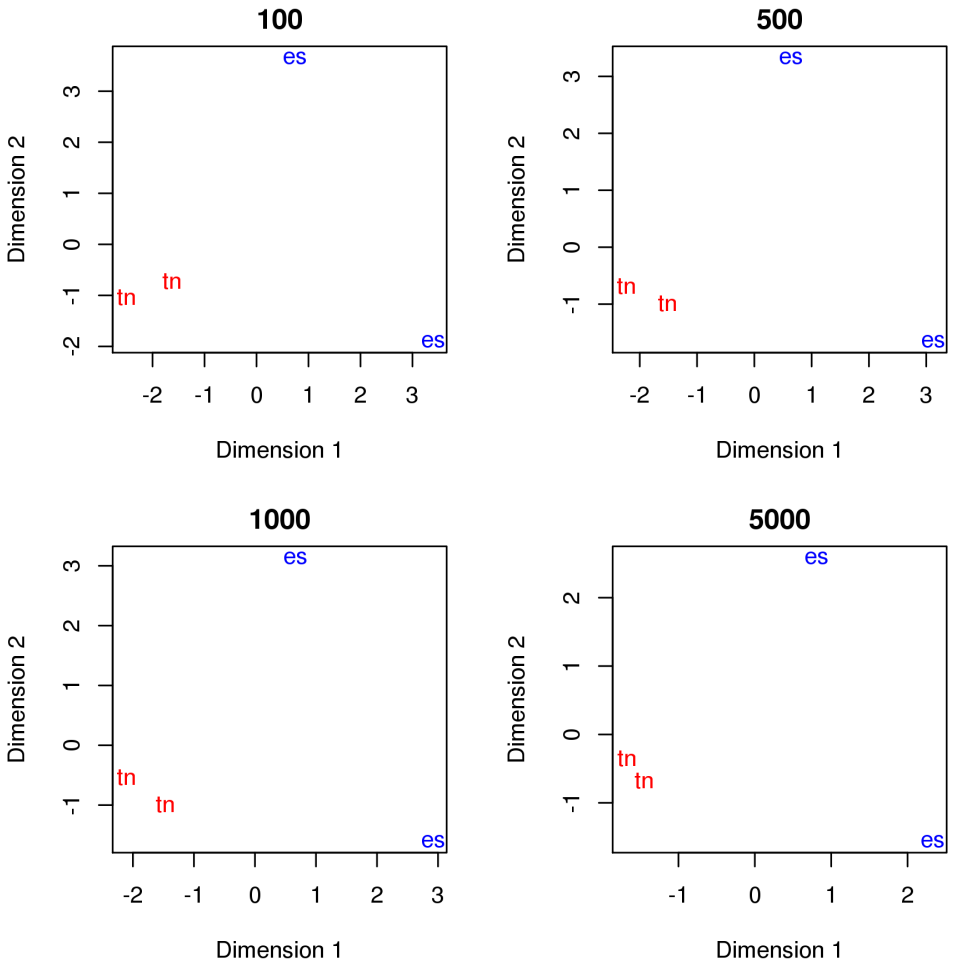
No normalization is performed to adjust for differences in mappability or sequencability between different regions of the genome. Region-specific biases are assumed to be constant between libraries. This is generally reasonable as the biases depend on fixed properties of the genome sequence such as GC content. Thus, biases should cancel out during DB comparisons. Any variability in the bias between samples will be absorbed into the dispersion estimate.

That said, explicit normalization to correct these biases can improve results for some datasets. Procedures like GC correction could decrease the observed variability by removing systematic differences between replicates. Of course, this also assumes that the targeted differences have no biological relevance. Detection power may be lost if this is not true. For example, differences in the GC content distribution can be driven by technical bias as well as biology, e.g., when protein binding is associated with a specific GC composition.

3.5 Examining replicate similarity with MDS plots

On a semi-related note, the binned counts can be used to examine the similarity of replicates through multi-dimensional scaling (MDS) plots. The distance between each pair of libraries is computed as the square root of the mean squared log-fold change across the top set of bins with the highest absolute log-fold changes. A small top set visualizes the most extreme differences whereas a large set visualizes overall differences. Again, binning is necessary as fold changes will be undefined in the presence of zero counts.

```
> par(mfrow=c(2,2), mar=c(5,4,2,2))
> binned <- windowCounts(bam.files, bin=TRUE, width=2000L)
> adj.counts <- cpm(binned$counts, log=TRUE)
> for (top in c(100, 500, 1000, 5000)) {
+     out <- plotMDS(adj.counts, main=top, col=c("blue", "blue", "red", "red"),
+         labels=c("es", "es", "tn", "tn"), top=top)
+ }
```



Replicates from different groups should form separate clusters in the plot. This indicates that the results are reproducible and that the effect sizes are large. Mixing between replicates of different conditions indicates that the biological difference has no effect on protein binding, or that the data is too variable for any effect to manifest. Any outliers should also be noted as their presence may confound the downstream analysis. In the worst case, the removal of the corresponding libraries may be necessary to obtain sensible results.

Chapter 4

Filtering prior to correction

This chapter will require `frag.len` and `data` defined in Chapter 2. We will also need the `normfacs` vector from Chapter 3. Oh, and the `me.demo` list as well, just for a demonstration at the end of this chapter.

4.1 Independent filtering for count data

Many of the low abundance windows in the genome correspond to background regions in which DB is not expected. Indeed, windows with low counts will not provide enough evidence against the null hypothesis to obtain sufficiently low p-values for DB detection. Similarly, some approximations used in the statistical analysis will fail at low counts. Removing such uninteresting or ineffective tests reduces the severity of the multiple testing correction, increases detection power amongst the remaining tests and reduces computational work.

Filtering is valid so long as it is independent of the test statistic under the null hypothesis [Bourgon et al., 2010]. In the negative binomial (NB) framework, this (probably) corresponds to filtering on the overall NB mean. The DB p-values retained after filtering should be uniform under the null hypothesis, assuming analogous behaviour to the normal case. Row sums can also be used for datasets where the effective library sizes are not very different or where the counts are assumed to be Poisson-distributed between biological replicates.

```
> abundances <- aveLogCPM(data$counts, lib.size=data$totals*normfacs)
> length(abundances)

[1] 473281

> summary(abundances)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.110  -2.029  -1.890  -1.660  -1.542  12.470
```

For demonstration purposes, an arbitrary threshold of -1 is used here to filter the window abundances. One can then restrict the dataset to the filtered values. While filtering can be performed at any stage of the analysis prior to the multiple testing correction, doing so at earlier steps is recommended to reduce computational work. Downstream estimates of various statistics are also more relevant when restricted to the regions of interest. Of course, one should retain enough points for information sharing in Chapter 5.

```
> keep <- abundances > -1
> demo <- data
> demo$counts <- demo$counts[keep,]
> demo$region <- demo$region[keep]
> sum(keep)
```

```
[1] 52369
```

The exact choice of filter threshold may not be immediately obvious. A filter that is too conservative will be ineffective whereas a filter that is too aggressive may reduce power by removing false nulls. In some respects, the filter value is necessarily arbitrary as it reflects prior expectations of the abundances of the features of interest. Nonetheless, several strategies for defining the filter threshold are described below.

4.2 By proportion

One approach is to to assume that only a certain proportion - say, 0.1% - of the genome is genuinely bound. The number of windows corresponding to these bound regions can be calculated as the proportion of the total number of windows, the latter of which can be derived from the genome length and the `spacing` interval used in `windowCounts`. The top set of windows with the highest abundances can then be selected such that the size of the retained set is equal to the computed number.

```
> spacing <- 50
> desired <- 0.001
> genome.windows <- sum(seqlengths(data$region)/spacing)
> keep <- length(abundances) - rank(abundances) + 1 < genome.windows*desired
> sum(keep)
```

```
[1] 54617
```

This approach is simple and has the practical advantage of maintaining a constant number of windows for the downstream analysis. However, it may not adapt well to different datasets where the proportion of bound sites can vary. Using an inappropriate percentage of binding sites will result in the loss of potential DB regions or inclusion of background regions.

4.3 By global enrichment

An alternative approach involves choosing a filter based on enrichment over the non-specific background. Specifically, the median (or any other robust average) of the binned abundances can be used as an estimate of the global background coverage in the dataset. Binning is necessary here to increase the size of the counts. This ensures that precision is maintained when estimating the average background abundance.

```
> bin.size <- 2000L
> binned <- windowCounts(bam.files, bin=TRUE, width=bin.size)
> bin.ab <- aveLogCPM(binned$counts, lib.size=binned$total*normfacs)
> threshold <- median(bin.ab)
```

As the threshold is computed for large bins, it needs to be scaled for a proper comparison to window abundances. If one assumes that reads are uniformly distributed in background regions, the threshold can be directly scaled down based on differences in the size of the read counting interval between windows and bins. For windows with read extension, the size of the interval is equal to `ext + width`. Only one `ext` is added because the expansion of the interval by read extension is strand-specific, i.e., only forward-mapping reads are counted on the left of the window and only reverse-mapping reads are counted on the right.

```
> width <- median(width(data$region))
> eff.win.size <- width + frag.len
> adjustment <- log2(bin.size/eff.win.size)
> threshold <- threshold - adjustment
```

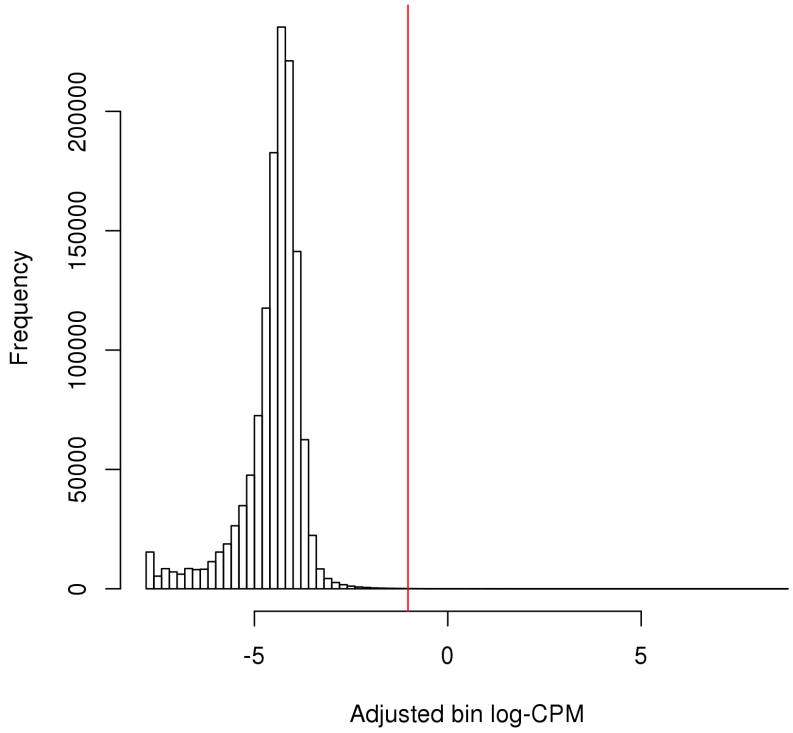
Windows are filtered based on some minimum required fold change over the global background. Here, a fold change of 10 over the background is necessary for a window to be considered as containing a binding site. This approach has an intuitive and experimentally relevant interpretation which adapts to the level of non-specific enrichment in the dataset.

```
> log.min.fc <- log2(10)
> threshold <- threshold+log.min.fc
> keep <- abundances >= threshold
> sum(keep)
```

```
[1] 54662
```

The effect of filtering can also be visualized with a histogram. This allows users to confirm that the bulk of (assumed) background windows are discarded upon filtering. Note that windows containing genuine binding sites will usually fail to appear on such plots due to the dominance of the background windows throughout the genome.

```
> hist(bin.ab-adjustment, xlab="Adjusted bin log-CPM",  breaks=100, main="")
> abline(v=threshold, col="red")
```



Of course, the pre-specified minimum fold change may be too aggressive when binding is weak. For TF data, a large cut-off works well as narrow binding sites will have high read densities and are unlikely to be lost during filtering. Smaller minimum fold changes are recommended for diffuse marks where the difference from background is less obvious.

4.4 By local enrichment

4.4.1 Mimicking single-sample peak callers

Local background estimators can also be constructed with a little imagination. This avoids inappropriate filtering when there are differences in background coverage across the genome. Here, the 2 kbp region surrounding each window will be used as the "neighbourhood" over which a local estimate of non-specific enrichment for that window can be obtained. The counts for this region can be obtained with the aptly-named `regionCounts` function.

```
> surrounds <- 2000
> neighbour <- suppressWarnings(resize(data$region,
+     width(data$region)+surrounds, fix="center"))
> wider <- regionCounts(bam.files, regions=neighbour, ext=frag.len)
```

Counts for each window are subtracted from the counts for its neighbourhood. This ensures that any enriched regions or binding sites inside the window do not interfere with estimation of its local background. Again, some adjustment for width is required for a valid comparison. Read extension is used for the neighbourhood counts, so note the addition of `frag.len`. The adjustment must also account for the subtraction of the window counts.

```
> neighbour.counts <- wider$counts - data$counts
> adjustment <- log2((surrounds + frag.len - eff.win.size)/eff.win.size)
```

Enrichment is defined as the fold change of the average window abundance over the local neighbourhood. Filtering can then be performed using a quantile- or fold change-based threshold on the enrichment values. This roughly mimics the behaviour of single-sample peak-calling programs such as MACS [Zhang et al., 2008].

```
> neighbour.ab <- aveLogCPM(neighbour.counts, lib.size=wider$totals*normfacs)-adjustment
> en.ab <- abundances - neighbour.ab
> summary(en.ab)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -9.524   1.445   1.930   1.872   2.330  12.340
```

Note that this procedure also assumes that no other enriched regions are present in each neighbourhood. Otherwise, the local background will be overestimated and windows may be incorrectly filtered out. This may be problematic for diffuse histone marks or TFBS clusters where enrichment may be observed in both the window and its neighbourhood.

If this seems too complicated, a simpler alternative is to identify locally enriched regions using peak-callers like MACS. Filtering can be performed to retain only windows within called peaks. However, peak calling must be done independently of the DB status of each window. If libraries are of similar size or biological variability is low, reads can be pooled into a single library for single-sample peak calling. This is equivalent to filtering on the average count and avoids distortion of the type I error rates due to data snooping.

4.4.2 With negative controls

Negative controls for ChIP-seq refer to input or IgG libraries where the IP step has been skipped or compromised, respectively. This accounts for sequencing/mapping biases in ChIP-seq data. IgG controls also quantify the amount of non-specific enrichment throughout the genome. These controls are mostly irrelevant when testing for DB between ChIP samples. However, they can be used to filter out windows with an average count in the ChIP sample below that of the control. The dummy example below requires a 5-fold or greater increase over the control to retain the window.

```
> in.demo <- windowCounts(c(bam.files, "IgG.bam"), ext=frag.len)
> chip <- aveLogCPM(in.demo$counts[,1:4], lib.size=in.demo$totals[1:4])
> control <- aveLogCPM(in.demo$counts[,5], lib.size=in.demo$totals[5])
> keep <- chip > control + log2(5)
```

It is worth mentioning that the csaw pipeline can also be applied to search for "DB" between ChIP libraries and control libraries. The ChIP and control libraries can be treated as separate groups, in which most "DB" events are expected to be enriched in the ChIP samples. If this is the case, the filtering procedure described above is inappropriate as it will select for windows with differences between ChIP and control samples. This compromises the assumption of the null hypothesis during testing, resulting in loss of type I error control.

4.5 By prior information

When only a subset of genomic regions are of interest, DB detection power can be improved by removing windows lying outside of these regions. Such regions could include promoters, enhancers, gene bodies or exons. The example below retrieves the coordinates of the broad gene bodies from the mouse genome, including the 3 kbp region upstream of the TSS that represents the putative promoter region for each gene.

```
> require(org.Mm.eg.db)
> suppressWarnings(anno <- select(org.Mm.eg.db, keys=keys(org.Mm.eg.db),
+     col=c("CHRLOC", "CHRLOCEND"), keytype="ENTREZID"))
> anno <- anno[!is.na(anno$CHRLOCCHR),]
> extension <- 3000
> coord5 <- ifelse(anno$CHRLOC > 0, anno$CHRLOC-extension, -anno$CHRLOC)
> coord3 <- ifelse(anno$CHRLOC > 0, anno$CHRLOCEND, -anno$CHRLOCEND+extension)
> broads <- GRanges(paste0("chr", anno$CHRLOCCHR), IRanges(coord5, coord3))
> head(broads)

GRanges with 6 ranges and 0 metadata columns:
      seqnames               ranges strand
         <Rle>            <IRanges>  <Rle>
  [1]     chr9 [ 21062393,  21070093]      *
  [2]     chr7 [ 84940169,  84967009]      *
  [3]    chr10 [ 77708457,  77712009]      *
  [4]    chr11 [ 45805083,  45842878]      *
  [5]     chr4 [144157556, 144165651]      *
  [6]     chr4 [134745412, 134771004]      *
  ---
  seqlengths:
                   chr9                 chr7 ...       chrUn_JH584304
                     NA                   NA ...                   NA
```

Windows can be filtered to only retain those which overlap with the regions of interest. Discerning users may wish to distinguish between full and partial overlaps, though this should not be a significant issue for small windows. This could also be combined with abundance filtering to retain windows that contain binding sites in the regions of interest.

```
> suppressWarnings(keep <- overlapsAny(data$region, broads))
> sum(keep)
```

```
[1] 202254
```

Any information used here should be independent of the DB status under the null in the current dataset. For example, DB calls from a separate dataset and/or independent annotation can be used without problems. However, using DB calls from the same dataset to filter regions would violate the null assumption and compromise type I error control.

4.6 Relationship between filtering and normalization

Note that the NB mean computed by `aveLogCPM` depends on the correct specification of the library sizes. In the example above, the effective library sizes after normalization are used after multiplying by `normfacs`. This ensures that composition biases are considered when computing the average count. In other situations, this may result in some circular dependencies when filtering is also required prior to normalization, e.g., normalization for efficiency biases in Section 3.2. Inquisitive users may wish to perform multiple iterations of filtering/normalization to ensure that the results are self-consistent.

```
> for (it in 1:3) {
+     ab <- aveLogCPM(me.demo$counts, lib.sizes=me.demo$total*me.norm)
+     keep <- rank(ab) > 0.99*length(ab)
+     me.norm <- normalizeChIP(me.demo$count[keep,], lib.sizes=me.demo$totals)
+     cat("Iteration is", it, "\n")
+     print(me.norm)
+ }

Iteration is 1
[1] 0.7108908 1.4066859
Iteration is 2
[1] 0.7108908 1.4066859
Iteration is 3
[1] 0.7108908 1.4066859
```

It is worth noting that the `aveLogCPM` function also depends on the estimated NB dispersion (see Chapter 5). However, dispersion estimation can only proceed after normalization and filtering. This results in another circular dependency that is resolved/ignored by having a "near enough is good enough" philosophy, i.e., using a sensible but otherwise arbitrary value for the NB dispersion in `aveLogCPM`. The alternative would be to iterate over the entire analysis which would be prohibitively time-consuming in most circumstances.

Chapter 5

Testing differential binding

For this next section, we'll be needing the `data` list from Chapter 2 and filtered in Chapter 4. Just let me assign the filtered list back to `data`, because I put it in a dummy variable:

```
> original <- data
> data <- demo
```

You'll also need the `normfacs` vector from Chapter 3, as well as the `design` matrix from the introduction.

5.1 Introduction to edgeR

5.1.1 Overview

Low counts per window are typically observed in ChIP-seq datasets, even for genuine binding sites. Any statistical analysis to identify DB sites must be able to handle discreteness in the data. Software packages using count-based models are ideal for this purpose. In this guide, the quasi-likelihood (QL) framework in the edgeR package is used [Lund et al., 2012]. Counts are modelled using NB distributions that account for overdispersion between biological replicates [Robinson and Smyth, 2008]. Each window can then be tested for significant differences between counts for different biological conditions.

It should be noted that any statistical method can be used if it is able to accept a count matrix and a vector of normalization factors (or more generally, a matrix of offsets). The choice of edgeR is primarily motivated by its performance relative to alternatives [Law et al., 2014], though the author's desire to increase his h-index is also a factor.

5.1.2 Setting up the data

First, a `DGEList` object must be formed from the count matrix. Additional information like the library size and the normalization factors should be included. For this analysis, the `normfacs` vector from TMM normalization of background bins is used. If an offset matrix is necessary (e.g., from non-linear normalization), this can be assigned into `y$offset` for later use in the various edgeR functions.

```
> y <- DGEList(data$counts, lib.size=data$totals, norm.factors=normfacs)
```

Observant readers will notice that the library size here and in all calls to `normalizeChIP` and `aveLogCPM` is set to `...$totals`. This is critical as it ensures consistency between normalization, filtering and the downstream statistical processing, i.e., the same library sizes should be used throughout all steps. The `totals` vector is used as it is constant for all window/bin sizes with any given dataset/read extraction parameters. Simply taking the column sums is inappropriate due to overlaps between windows and the effects of filtering.

The experimental design is described by a design matrix. In this case, the only relevant factor is the cell type of each sample. A generalized linear model (GLM) will be fitted to the counts for each window using the specified design matrix [McCarthy et al., 2012]. This provides a general framework for the analysis of complex experiments with multiple factors. Readers are referred to the user's guide in edgeR for more details on parametrization.

```
> design

  intercept cell.type
1         1         0
2         1         0
3         1         1
4         1         1
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$`factor(c("es", "es", "tn", "tn"))`
[1] "contr.treatment"
```

5.2 Estimating the dispersions

5.2.1 Stabilising estimates with empirical Bayes

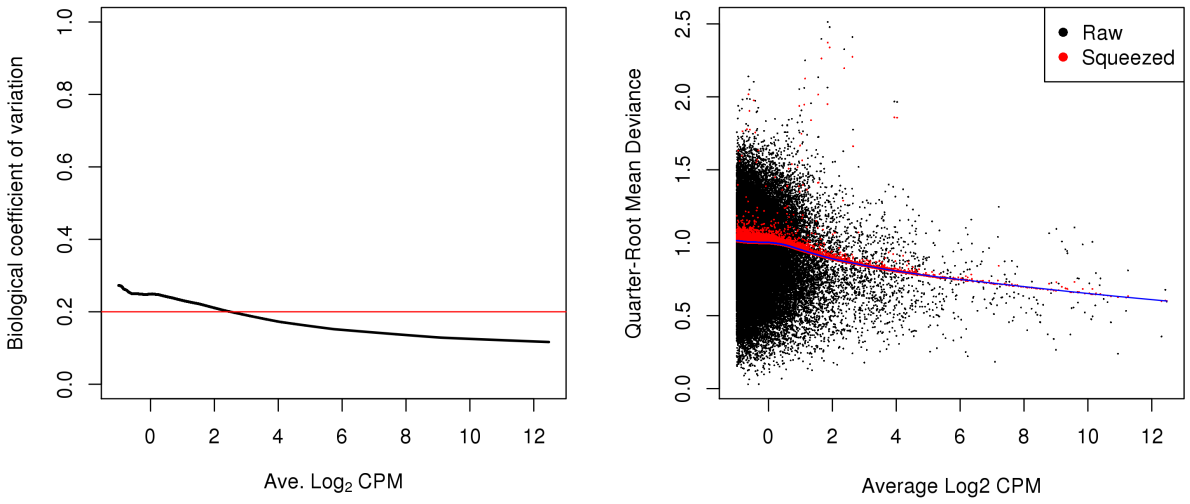
Under the QL framework, both the QL and NB dispersions are used to model biological variability in the data [Lund et al., 2012]. The former ensures that the NB mean-variance relationship is properly specified with appropriate contributions from the Poisson and Gamma components. The latter accounts for variability and uncertainty in the dispersion estimate.

However, limited replication in most ChIP-seq experiments means that each window does not contain enough information for precise estimation of either dispersion.

This problem is overcome in edgeR by sharing information across windows. For the NB dispersions, a mean-dispersion trend is fitted across all windows to model the mean-variance relationship [McCarthy et al., 2012]. The raw QL dispersion for each window is estimated after fitting a GLM with the trended NB dispersion. Another mean-dependent trend is fitted to the raw QL estimates. An empirical Bayes (EB) strategy is then used to stabilize the raw QL dispersion estimates by shrinking them towards the second trend [Lund et al., 2012]. The ideal amount of shrinkage is determined from the heteroskedasticity of the data.

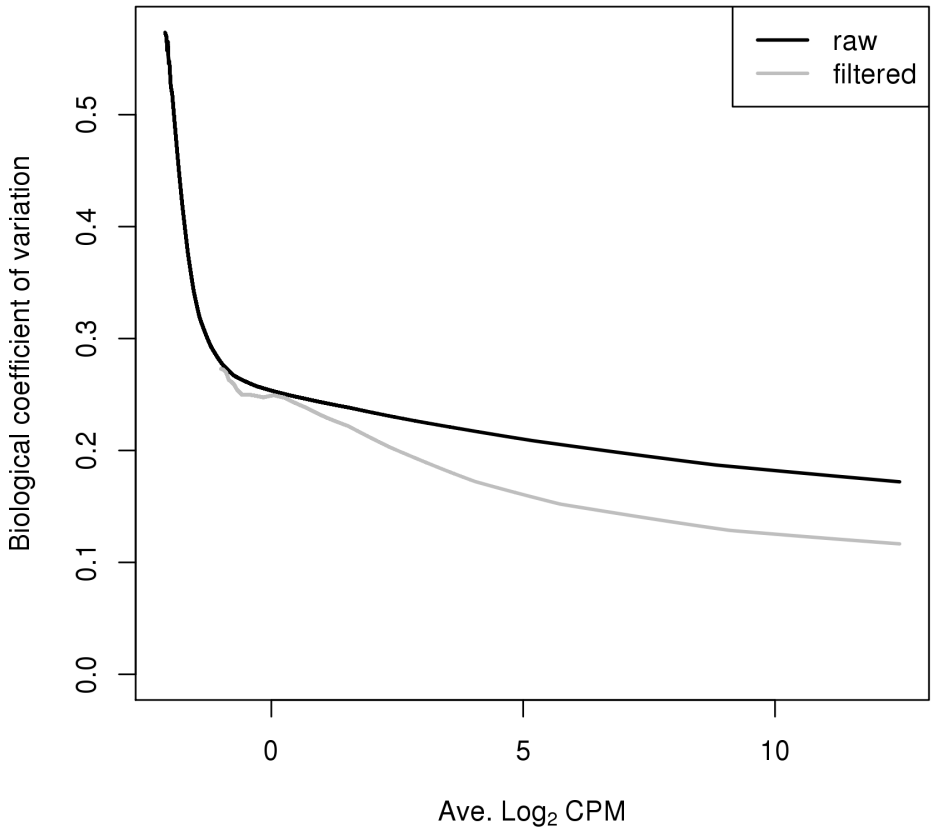
```
> par(mfrow=c(1,2))
> y <- estimateDisp(y, design)
> o <- order(y$AveLogCPM)
> plot(y$AveLogCPM[o], sqrt(y$trended.dispersion[o]), type="l", lwd=2,
+   ylim=c(0, 1), xlab=expression("Ave."~Log[2]~"CPM"),
+   ylab=("Biological coefficient of variation"))
> abline(h=0.2, col="red")
> results <- glmQLFTest(y, design, robust=TRUE, plot=TRUE)
```

The effect of EB stabilisation can be visualized by examining the biological coefficient of variation (for the NB dispersion) and the quarter-root deviance (for the QL dispersion). These plots can also be used to decide whether the fitted trend is appropriate. Sudden irregularities may be indicative of an underlying structure in the data which cannot be modelled with the mean-dispersion trend. Discrete patterns in the raw dispersions are indicative of low counts and suggest that more aggressive filtering is required.



A strong trend may also be observed where the dispersion drops sharply with increasing mean. This is due to the disproportionate impact of artifacts such as mapping errors and PCR duplicates at low counts. It is difficult to accurately fit an empirical curve to these strong trends. Inaccurate fitting means that the dispersions at high abundances are often overestimated. Users should check whether removal of the low abundance regions affects the dispersion estimate. Large changes upon removal imply that more aggressive abundance-based filtering may be desirable, as described in Chapter 4.

```
> yo <- DGEList(original$counts, lib=original$total, norm=normfacs)
> yo <- estimateDisp(yo, design)
> oo <- order(yo$AveLogCPM)
> plot(yo$AveLogCPM[oo], sqrt(yo$trended.dispersion[oo]), type="l", lwd=2,
+   ylim=c(0, max(sqrt(yo$trended))), xlab=expression("Ave."~Log[2]~"CPM"),
+   ylab=("Biological coefficient of variation"))
> lines(y$AveLogCPM[o], sqrt(y$trended[o]), lwd=2, col="grey")
> legend("topright", c("raw", "filtered"), col=c("black", "grey"), lwd=2)
```



5.2.2 Modelling heteroskedasticity

The heteroskedasticity of the data is modelled in edgeR by the prior degrees of freedom (d.f.). A large value for the prior d.f. indicates that heteroskedasticity is low. This means that more EB shrinkage can be performed to reduce uncertainty and maximize power. However, strong shrinkage is not appropriate if the dispersions are highly variable. Fewer prior degrees of freedom (and less shrinkage) is required to maintain type I error control.

```
> summary(results$df.prior)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5323 63.7300 63.7300 62.6000 63.7300 63.7300
```

On occasion, the estimated prior degrees of freedom will be infinite. This is indicative of a strong batch effect where the dispersions are large with minimal variability. A typical example would involve uncorrected differences in IP efficiency across replicates. In severe cases, the trended dispersion may fail to pass through the bulk of points as the variability is too low to be properly modelled in the QL framework. These batch effects are often caused by efficiency biases and can be removed with appropriate normalization.

It is worth pointing out that the prior degrees of freedom should be robustly estimated [Phipson et al., 2013]. Obviously, this protects against large positive outliers (e.g. highly variable windows) but it also protects against near-zero dispersions at low counts. These will manifest as large negative outliers after a log transformation step during estimation [Smyth, 2004]. Without robust estimation, incorporation of these outliers will inflate the observed variability in the dispersions and reduce DB detection power.

5.3 Testing for DB windows

The effect of specific factors can be tested to identify windows with significant differential binding. In the QL framework, p-values are computed using the F-test [Lund et al., 2012]. This is more appropriate than using the likelihood ratio test as the F-test accounts for uncertainty in the dispersion estimates. Associated statistics such as log-fold changes and log-counts per million are also computed for each window.

```
> results <- glmQLFTest(y, design, robust=TRUE, contrast=c(0, 1))
> head(results$table)

       logFC     logCPM         F     PValue
1 -0.5078927 -0.5402696 1.0563315 0.30782063
2 -0.4598138  0.4265961 1.2652509 0.27607407
3 -0.5350752 -0.3695510 1.2712313 0.26363466
4 -0.4692340 -0.4043727 0.9678305 0.32882914
5 -0.8460653 -0.4803792 2.9451508 0.09084544
6 -0.2987352 -0.5264553 0.3757897 0.54198044
```

The null hypothesis here is that the cell type has no effect. The `contrast` argument in the `glmQLFTest` function specifies which factors are of interest. In this case, a contrast of `c(0, 1)` defines the null hypothesis as `0*intercept + 1*cell.type = 0`, i.e., that the log-fold change between cell types is zero. DB windows can then be identified by rejecting the null. Specification of the contrast is explained in greater depth in the edgeR user's manual.

As a side note, the `glmQLFTest` function will perform both the dispersion estimation and the hypothesis testing. As such, it only needs to be called once. Multiple calls are only shown here and in Section 5.2.1 for demonstration purposes.

Chapter 6

Correction for multiple testing

All right, we're almost there. This chapter needs the `results` object from the last chapter. You'll also need the filtered `data` list from Chapter 2, as well as the `broads` object from Chapter 4.

6.1 Problems with false discovery rate control

The false discovery rate (FDR) is usually the most appropriate measure of error for high-throughput experiments. Control of the FDR can be provided by applying the Benjamini-Hochberg (BH) method [Benjamini and Hochberg, 1995] to a set of p-values. This is less conservative than the alternatives (e.g., Bonferroni) yet still provides some measure of error control. The most obvious approach is to apply the BH method to the set of p-values across all windows. This will control the FDR across the set of putative DB windows.

However, the FDR across all detected windows is not necessarily the most relevant error rate. Interpretation of ChIP-seq experiments is more concerned with regions of the genome in which (differential) protein binding is found, rather than the individual windows. In other words, the FDR across all detected DB regions is usually desired. This is not equivalent to that across all DB windows as each region will often consist of multiple overlapping windows. Control of one will not guarantee control of the other [Lun and Smyth, 2014].

To illustrate this difference, consider an analysis where the FDR across all window positions is controlled at 10%. In the results, there are 18 adjacent window positions forming one cluster and 2 windows forming a separate cluster. Each cluster represents a region. The first set of windows is a truly DB region whereas the second set is a false positive. A window-based interpretation of the FDR is correct as only 2 of the 20 window positions are false positives. However, a region-based interpretation results in an actual FDR of 50%.

Problems from misinterpretation can be avoided by applying the BH method to a p-value

from each region. Windows can be clustered together into regions with a number of strategies. Simes' method can then be used to compute a combined p-value for each cluster based on the p-values the constituent windows [Simes, 1986]. This tests the joint null hypothesis that no enrichment is observed across any sites within the region. The combined p-values are then adjusted using the BH method to control the region-level FDR.

6.2 Clustering with external information

Combined p-values can be computed for a pre-defined set of regions based on the windows overlapping those regions. The most obvious source of pre-defined regions is that of annotated features such as promoters or gene bodies. Alternatively, called peaks can be used provided that sufficient care has been taken to avoid loss of error control from data snooping. In either case, the `findOverlaps` function from the GenomicRanges package can be used to identify all windows in or overlapping each specified region.

```
> olap <- findOverlaps(broads, data$region)
> olap

Hits of length 26071
queryLength: 27294
subjectLength: 52369
      queryHits subjectHits
       <integer>   <integer>
 1             7       36290
 2            11       48150
 3            18       39194
 4            18       39195
 5            18       39196
 ...         ...         ...
 26067     27283       35634
 26068     27283       35635
 26069     27286       36047
 26070     27286       36048
 26071     27286       36049
```

The `combineTests` function can then be used to combine the p-values for all windows in each region. This provides a single combined p-value (and its FDR-adjusted value) for each region. The row names of the output table correspond to the value of the cluster identifiers supplied in `ids`. These should, in turn, act as indices for the regions of interest in `broads`. The average log-CPM and log-FC across all windows in each region are also computed.

```
> cluster.ids <- queryHits(olap)
> window.ids <- subjectHits(olap)
> tabprom <- combineTests(cluster.ids, results$table[window.ids,,drop=FALSE])
> head(tabprom)
```

```
        logFC     logCPM       PValue          FDR
7  2.95211689 -0.7893144 1.987679e-05 0.0002377056
11 0.95217836 -0.8375883 8.860506e-02 0.1451496065
18 1.56736941  0.1369136 9.613185e-04 0.0043771167
22 0.98826601 -0.9671633 9.477974e-02 0.1536700621
23 1.10365009 -0.2706485 4.965306e-02 0.0916712846
25 0.08376962 -0.4331375 9.041186e-01 0.9346915676
```

At this point, one might imagine that it would be simpler to just collect and analyze counts over the predefined regions. This is a valid strategy but will yield different results. Consider a promoter containing two separate peaks which are identically DB in opposite directions. Counting reads across the promoter will give equal counts for each group so changes within the promoter will not be detected. For peaks, imprecise boundaries for the called peaks can lead to loss of DB detection power due to "contamination" by reads from background regions. In both cases, window-based methods may be more robust as each interval of the promoter/peak region is examined separately [Lun and Smyth, 2014].

6.3 Quick and dirty clustering

Clustering can also be quickly performed inside csaw with a simple single-linkage algorithm, implemented in the `mergeWindows` function. This approach can be useful as it avoids potential problems with the other clustering methods, e.g., peak-calling errors, incorrect or incomplete annotation. Briefly, all windows which are less than some distance apart - say, 1 kbp - are put in the same cluster. This reflects some arbitrary minimum distance at which two binding events are considered to be separate sites.

```
> merged <- mergeWindows(data$region, tol=1000L)
> merged$region
```

```
GRanges with 13665 ranges and 0 metadata columns:
          seqnames               ranges strand
             <Rle>            <IRanges>  <Rle>
      [1]     chr1   [3695601, 3695760]      *
      [2]     chr1   [3981801, 3981810]      *
      [3]     chr1   [5072001, 5072010]      *
      [4]     chr1   [7397901, 7398110]      *
      [5]     chr1   [7860851, 7860860]      *
      ...      ...                 ...    ...
  [13661]     chrY [90782851, 90782910]      *
  [13662]     chrY [90784051, 90784110]      *
  [13663]     chrY [90805151, 90805210]      *
  [13664]     chrY [90808851, 90808860]      *
  [13665]     chrY [90812101, 90813560]      *
  ---
  seqlengths:
                    chr1                chr10 ... chrY_JH584303_random
               195471971            130694993 ...                158099
```

A combined p-value is computed for each cluster as previously described. Application of the BH method controls the FDR across all detected clusters. Like before, the row names in the output table are indices for the corresponding coordinates of the clusters in `merged$regions`. This allows for simple correspondence between the results and the regions.

```
> tabcom <- combineTests(merged$id, results$table)
> head(tabcom)

       logFC     logCPM     PValue        FDR
1 -0.4930039 -0.2218993 0.32882914 0.58915041
2 -0.8460653 -0.4803792 0.09084544 0.26014312
3 -0.2987352 -0.5264553 0.54198044 0.77730506
4  1.0222183  0.4770579 0.01119654 0.05846416
5 -1.0761098 -0.9931622 0.06697002 0.21188824
6  1.3899478 -0.5090193 0.01215433 0.06215900
```

If many overlapping windows are present, very large clusters may be formed that are difficult to interpret. A simple check can be used to determine whether most clusters are of an acceptable size. Many huge clusters indicate that more aggressive filtering from Chapter 4 is required. This mitigates chaining effects by reducing the density of windows in the genome.

```
> summary(width(merged$region))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     10      10      60     269     160   51410
```

Alternatively, chaining can be limited by setting the `max.width` parameter to restrict the sizes of the merged intervals. The chosen value should be small enough so as to be separate differentially bound regions from unchanged neighbours, yet large enough to avoid difficulties in interpretation from many adjacent windows. A value from 2000 to 10000 bp is recommended. This can viewed as the maximum distance at which two binding sites are considered part of the same event.

```
> merged <- mergeWindows(data$region, tol=1000L, max.width=5000L)
> summary(width(merged$region))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.0    10.0    60.0   263.8   160.0  4960.0
```

There are also provisions for clustering based on the sign of the log-fold change. The idea is that clusters will be broken up wherever the sign changes. This will separate binding sites that are close together but are changing in opposite directions. A vector can be supplied in `sign` to indicate whether each window has a positive log-fold change.

```
> merged <- mergeWindows(data$region, tol=1000L, sign=(results$table$logFC > 0))
> summary(width(merged$region))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.0    10.0    60.0   166.8   110.0 49860.0
```

However, sign-based filtering is not recommended as the sign of windows within each cluster is not independent of their DB status. Windows in a genuine DB region will form one cluster (consistent sign) whereas those in non-DB regions will form many clusters (inconsistent sign, as the log-fold change is small). This results in conservativeness as more clusters will have large p-values. Furthermore, any attempt to filter away small clusters will cause liberalness if too many large p-values are lost.

Chapter 7

Post-processing steps

This is where we bring it all together. We'll need the `merged` list and the `tabcom` table from the previous chapter. There's a bit about visualization at the end where we need the `y` object from Chapter 5 and the `bam.files` that we started off with. Oh, and the `org.Mm.eg.db` object that we loaded in Chapter 4.

7.1 Adding gene-based annotation

Annotation can be added to a given set of regions using the `detailRanges` function. This will identify overlaps between the regions and annotated features such as exons, introns and promoters. Here, the promoter region of each gene is defined as some interval 3 kbp up- and 1 kbp downstream of the TSS for that gene. Any exonic features within `dist` on the left or right side of the region will also be reported.

```
> require(TxDb.Mmusculus.UCSC.mm10.knownGene)
> sig.bins <- merged$region[as.integer(rownames(tabcom))]
> anno <- detailRanges(sig.bins, txdb=TxDb.Mmusculus.UCSC.mm10.knownGene,
+     orgdb=org.Mm.eg.db, promoter=c(3000, 1000), dist=5000)
> head(anno$overlap)

[1] ""          ""          "Rgs20|0|-" ""          ""          ""

> head(anno$left)

[1] ""                ""                "Rgs20|1|-[1716]" ""
[5] ""                ""

> head(anno$right)
```

```
[1] ""              ""              ""              ""
[5] ""              "Rrs1|1|+[3898]"
```

Character vectors of compact string representations are provided that describe the features overlapped by each supplied region. Each pattern represents `GENE|EXONS|STRAND` for the overlapped exons of that gene. Promoters are labelled as exon `0` whereas introns are labelled as `I`. For `left` and `right`, an additional `DISTANCE` field is included which indicates the gap between the annotated feature and the supplied region.

While the string representation is saves space, it is not easy to work with. If the annotation needs to manipulated directly, users can obtain it from the `detailRanges` command by not specifying the regions of interest. This can then be used for overlapping, e.g., to identify the genes containing DB sites overlapping the promoter.

```
> anno.ranges <- detailRanges(txdb=TxDb.Mmusculus.UCSC.mm10.knownGene, orgdb=org.Mm.eg.db)
> head(anno.ranges)

GRanges with 6 ranges and 2 metadata columns:
            seqnames               ranges strand |      symbol      exon
               <Rle>            <IRanges>  <Rle> | <character> <integer>
  100009600     chr9 [21062393, 21062717]      - |       Zglp1         5
  100009600     chr9 [21062894, 21062987]      - |       Zglp1         4
  100009600     chr9 [21063314, 21063396]      - |       Zglp1         3
  100009600     chr9 [21066024, 21066377]      - |       Zglp1         2
  100009600     chr9 [21066940, 21067925]      - |       Zglp1         1
  100009609     chr7 [84940169, 84941088]      - |     Vmn2r65         6
  ---
  seqlengths:
                   chr1                 chr2 ...       chrUn_JH584304
              195471971            182113224 ...               114452
```

7.2 Saving the results to file

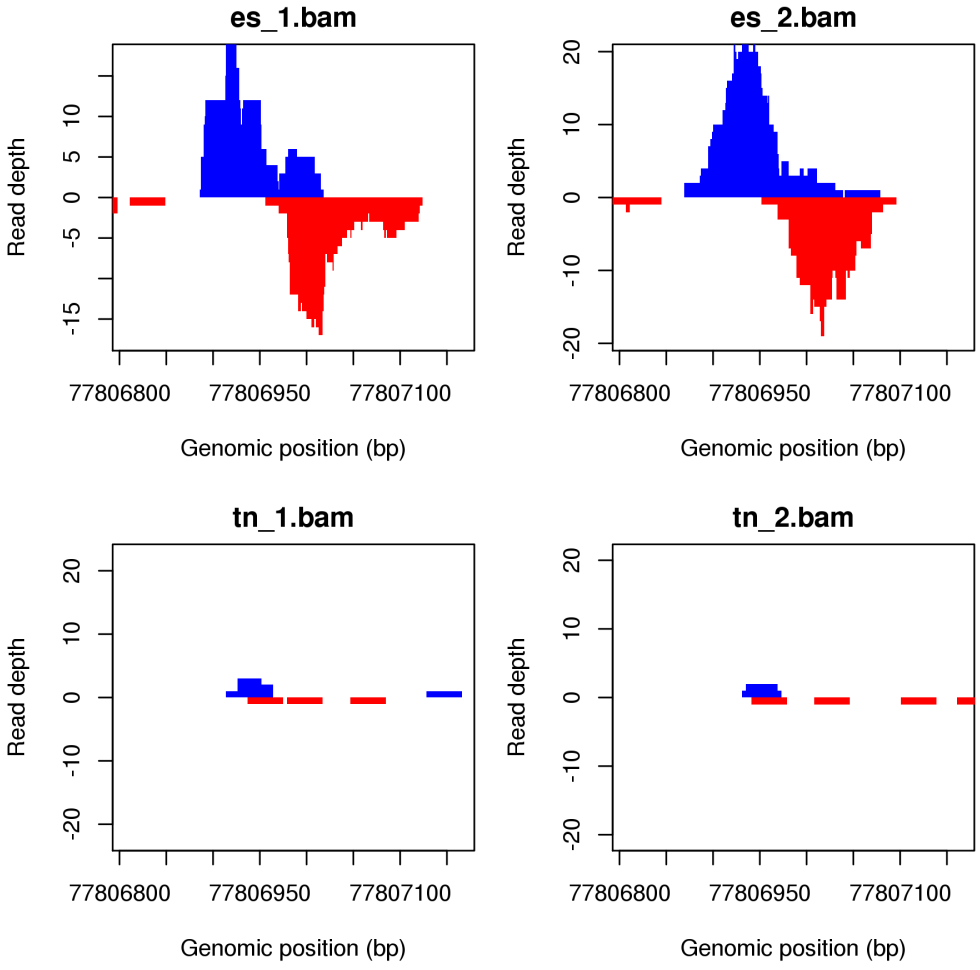
It is a simple matter to save the results for later perusal. This is done here in the `*.tsv` format where all detail is preserved. Compression is used to reduce the file size. Of course, other formats can be used depending on the purpose of the file, e.g., exporting to BED files through the rtracklayer package for visual inspection of the data with genomic browsers.

```
> ofile <- gzfile("clusters.gz", open="w")
> write.table(data.frame(chr=as.character(seqnames(sig.bins)), start=start(sig.bins),
+     end=end(sig.bins), tabcom, anno), file=ofile,
+     row.names=FALSE, quote=FALSE, sep="\t")
> close(ofile)
```

7.3 Simple home-made visualization

A quick and dirty inspection of the read depth around interesting features is also possible within csaw. The blue and red tracks represent the coverage on the forward and reverse strands, respectively. The height of each plot is adjusted according to the library sizes to avoid any misrepresentation of read depth.

```
> cur.region <- GRanges("chr18", IRanges(77806807, 77807165))
> lib.sizes <- exp(getOffset(y))
> mean.lib <- mean(lib.sizes)
> par(mfrow=c(2,2), mar=c(5, 4, 2, 1))
> for (i in 1:length(bam.files)) {
+     plotRegion(cur.region, bam.files[i],
+         max.depth=20*lib.sizes[i]/mean.lib, main=bam.files[i])
+ }
```



While there are definitely more sophisticated (and prettier) visualization methods available, the `plotRegion` function is provided as it uses the same read extraction parameters (e.g., `minq`, `dedup`) as `windowCounts`. Thus, it can be used to ensure that the visualized data is consistent with the analysis.

Chapter 8

Epilogue

Congratulations on getting to the end. Here's a poem for your efforts.

> There once was a man named Will
> Who never ate less than his fill.
> He ate meat and bread
> Until he was fed
> But died when he saw the bill.

8.1 Datasets

8.1.1 Obtaining the FastQ files

The main NFYA dataset used throughout the guide was first mentioned in Section 1.3. This was generated by Tiwari et al. [2012] and is available from the NCBI Gene Expression Omnibus (GEO) with the accession number GSE25532. FastQ files can be obtained from the Sequence Read Archive (SRA) with accession numbers of SRR074398 for `es_1.bam`, SRR074399 for `es_2.bam`, SRR074417 for `tn_1.bam` and SRR074418 for `tn_2.bam`.

The paired-end dataset used in Section 2.3 was generated by Pal et al. [2013] and is available from the NCBI GEO under the accession GSE43212. FastQ files can be obtained from the SRA with the accession SRR642390 for `example-pet.bam`.

All libraries used in Section 2.4 were generated by Zhang et al. [2012] and are available from the NCBI GEO under the accession GSE31233. FastQ files can be obtained from the SRA under the accessions SRR330784 and SRR330785 for `h3ac.bam`; SRR330800 and SRR330801 for `h3k4me2.bam`; and SRR330814, SRR330815 and SRR330816 for `h3k27me3.bam`. Each set of FastQ files represents technical replicates that are merged into a single BAM file.

Finally, the H3K4me3 dataset in Section 3.2 was generated by Revilla-I-Domingo et al.

[2012] and is available from the NCBI GEO under the accession GSE38046. FastQ files can be obtained from the SRA under the accessions SRR499732 and SRR499733 for `h3k4me3_pro.bam`, and SRR499716 and SRR499717 for `h3k4me3_mat.bam`. Again, each set of FastQ files represents technical replicates. For H3ac normalization, the FastQ file at SRR330786 was also downloaded and used as `h3ac_2.bam`.

8.1.2 Alignment and processing to produce BAM files

Technically, the libraries are all downloaded in the SRA format. These can be unpacked to yield FastQ files using the `fastq-dump` command from the SRA Toolkit (`http://eutils.ncbi.nih.gov/Traces/sra/?view=software`). For the paired-end library, users will need to specify `fastq-dump -split-files` to ensure that two separate files are produced, i.e., one containing sequences from each end.

The reads in the FastQ files can then be mapped. For this particular guide, all reads were aligned to the mm10 build of the mouse genome using the subread program [Liao et al., 2013]. This can be obtained from Bioconductor as Rsubread or as a standalone C program from `http://subread.sourceforge.net`. Default settings were used with the exception of the consensus threshold, which was lowered to 2 to accommodate the short read lengths. Paired-end data was aligned by supplying both FastQ files to subread in the same run.

Once aligned, SAM files were converted to BAM files using the samtools suite [Li et al., 2009]. BAM files were position-sorted with the `samtools sort` command, and duplicate reads were marked using the `MarkDuplicates` command from the Picard suite (`http://picard.sourceforge.net`). Any technical replicates were merged together using `samtools merge` to form a single library. Indexing was performed using `samtools index`.

8.2 Session information

```
> sessionInfo()

R version 3.1.0 (2014-04-10)
Platform: x86_64-unknown-linux-gnu (64-bit)

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=en_US.UTF-8    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] splines   parallel  stats     graphics  grDevices utils     datasets
[8] methods   base
```

```
other attached packages:
 [1] TxDb.Mmusculus.UCSC.mm10.knownGene_2.14.0
 [2] GenomicFeatures_1.16.2
 [3] org.Mm.eg.db_2.14.0
 [4] RSQLite_0.11.4
 [5] DBI_0.2-7
 [6] AnnotationDbi_1.26.0
 [7] Biobase_2.24.0
 [8] statmod_1.4.20
 [9] locfit_1.5-9.1
[10] edgeR_3.6.4
[11] limma_3.20.8
[12] csaw_0.99.0
[13] GenomicRanges_1.16.3
[14] GenomeInfoDb_1.0.2
[15] IRanges_1.22.9
[16] BiocGenerics_0.10.0

loaded via a namespace (and not attached):
 [1] BatchJobs_1.2           BBmisc_1.7              BiocParallel_0.6.1
 [4] biomaRt_2.20.0          Biostrings_2.32.0       bitops_1.0-6
 [7] brew_1.0-6              BSgenome_1.32.0         checkmate_1.0
[10] codetools_0.2-8         digest_0.6.4            fail_1.2
[13] foreach_1.4.2           GenomicAlignments_1.0.1 grid_3.1.0
[16] iterators_1.0.7         KernSmooth_2.23-12      lattice_0.20-29
[19] plyr_1.8.1              Rcpp_0.11.2             RCurl_1.95-4.1
[22] Rsamtools_1.16.1        rtracklayer_1.24.2      sendmailR_1.1-2
[25] stats4_3.1.0            stringr_0.6.2           tools_3.1.0
[28] XML_3.98-1.1            XVector_0.4.0           zlibbioc_1.10.0
```

8.3 References

K. V. Ballman, D. E. Grill, A. L. Oberg, and T. M. Therneau. Faster cyclic loess: normalizing RNA arrays via linear models. *Bioinformatics*, 20(16):2778–2786, Nov 2004.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. R. Stat. Soc. Series B*, 57:289–300, 1995.

R. Bourgon, R. Gentleman, and W. Huber. Independent filtering increases detection power for high-throughput experiments. *Proc. Natl. Acad. Sci. U.S.A.*, 107(21):9546–9551, May 2010.

P. Humburg, C. A. Helliwell, D. Bulger, and G. Stone. ChIPseqR: analysis of ChIP-seq experiments. *BMC Bioinformatics*, 12:39, 2011.

P. V. Kharchenko, M. Y. Tolstorukov, and P. J. Park. Design and analysis of ChIP-seq experiments for DNA-binding proteins. *Nat. Biotechnol.*, 26(12):1351–1359, Dec 2008.

S. G. Landt, G. K. Marinov, A. Kundaje, P. Kheradpour, F. Pauli, S. Batzoglou, B. E. Bernstein, P. Bickel, J. B. Brown, P. Cayting, Y. Chen, G. Desalvo, C. Epstein, K. I. Fisher-Aylor, G. Euskirchen, M. Gerstein, J. Gertz, A. J. Hartemink, M. M. Hoffman, V. R. Iyer, Y. L. Jung, S. Karmakar, M. Kellis, P. V. Kharchenko, Q. Li, T. Liu, X. S. Liu, L. Ma, A. Milosavljevic, R. M. Myers, P. J. Park, M. J. Pazin, M. D. Perry, D. Raha, T. E. Reddy, J. Rozowsky, N. Shoresh, A. Sidow, M. Slattery, J. A. Stamatoyannopoulos, M. Y. Tolstorukov, K. P. White, S. Xi, P. J. Farnham, J. D. Lieb, B. J. Wold, and M. Snyder. ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia. *Genome Res.*, 22(9):1813–1831, Sep 2012.

C. W. Law, Y. Chen, W. Shi, and G. K. Smyth. Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biol.*, 15(2):R29, Feb 2014.

H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25 (16):2078–2079, Aug 2009.

Y. Liao, G. K. Smyth, and W. Shi. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Res.*, 41(10):e108, May 2013.

A. T. Lun and G. K. Smyth. De novo detection of differentially bound regions for ChIP-seq data using peaks and windows: controlling error rates correctly. *Nucleic Acids Res.*, 42 (11):e95, Jul 2014.

S. P. Lund, D. Nettleton, D. J. McCarthy, and G. K. Smyth. Detecting differential expression in RNA-sequence data using quasi-likelihood with shrunken dispersion estimates. *Stat. Appl. Genet. Mol. Biol.*, 11(5), 2012.

D. J. McCarthy, Y. Chen, and G. K. Smyth. Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Res.*, 40(10): 4288–4297, May 2012.

B. Pal, T. Bouras, W. Shi, F. Vaillant, J. M. Sheridan, N. Fu, K. Breslin, K. Jiang, M. E. Ritchie, M. Young, G. J. Lindeman, G. K. Smyth, and J. E. Visvader. Global changes in the mammary epigenome are induced by hormonal cues and coordinated by Ezh2. *Cell Rep.*, 3(2):411–426, Feb 2013.

B. Phipson, S. Lee, I. J. Majewski, W. S. Alexander, and G. K. Smyth. Empirical Bayes in the presence of exceptional cases, with application to microarray data. Technical report, Bioinformatics Division, Walter and Eliza Hall Institute of Medical Research, 2013.

R. Revilla-I-Domingo, I. Bilic, B. Vilagos, H. Tagoh, A. Ebert, I. M. Tamir, L. Smeenk, J. Trupke, A. Sommer, M. Jaritz, and M. Busslinger. The B-cell identity factor Pax5 regulates distinct transcriptional programmes in early and late B lymphopoiesis. *EMBO J.*, 31(14):3130–3146, 2012.

M. D. Robinson and A. Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biol.*, 11(3):R25, 2010.

M. D. Robinson and G. K. Smyth. Small-sample estimation of negative binomial dispersion, with applications to SAGE data. *Biostatistics*, 9(2):321–332, Apr 2008.

M. D. Robinson, D. J. McCarthy, and G. K. Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, Jan 2010.

R. J. Simes. An improved Bonferroni procedure for multiple tests of significance. *Biometrika*, 73(3):751–754, 1986.

G. K. Smyth. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat. Appl. Genet. Mol. Biol.*, 3:Article3, 2004.

V. K. Tiwari, M. B. Stadler, C. Wirbelauer, R. Paro, D. Schubeler, and C. Beisel. A chromatin-modifying function of JNK during stem cell differentiation. *Nat. Genet.*, 44(1): 94–100, Jan 2012.

J. A. Zhang, A. Mortazavi, B. A. Williams, B. J. Wold, and E. V. Rothenberg. Dynamic transformations of genome-wide epigenetic marking and transcriptional control establish T cell identity. *Cell*, 149(2):467–482, Apr 2012.

Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, and X. S. Liu. Model-based analysis of ChIP-Seq (MACS). *Genome Biol.*, 9(9):R137, 2008.