csaw: ChIP-seq analysis with windows

User's Guide

Aaron Lun

First edition 15 August 2012

Last revised 10 October 2013

Contents

Chapter 1

Introduction

1.1 Scope

This document gives an overview of the Bioconductor package csaw for detecting differential binding 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. Statistical methods are based upon those in the edgeR package [Robinson et al., 2010]. As such, the user's guide for edgeR is recommended reading for a deeper understanding of the nuts and bolts of both packages.

1.2 How to get help

Most questions about csaw will hopefully be answered by the documentation or references. 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 read by typing `?windowCounts` or `help(windowCounts)` at the R prompt.

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 helpful 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 BAM file paths is supplied in `bam.files` and a design matrix is supplied in `design`.

```
> data <- windowCounts(bam.files, ext=110)
> y <- DGEList(data$counts, lib.size=data$totals)
> binned <- windowCounts(bam.files, bin=10000)
> y$samples$norm.factors <- normalizeChIP(binned$counts)
> y <- estimateDisp(y, design)
> results <- glmQLFTest(y, design, robust=TRUE)
> merged <- mergeWindows(data$region, tol=1000L)
> tabcom <- combineFDR(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 tests.
4. Identifying differentially bound (DB) windows.
5. Correcting for multiple hypothesis tests.

The use of the pipeline is demonstrated here using publicly available ChIP-seq data. The aim is to identify changes in NFYA binding between embryonic stem cells and terminal neurons [Tiwari et al., 2012]. Reads in these libraries were mapped to the mm10 build of the mouse genome using Rsubread [Liao et al., 2013] and sorted using SAMtools [Li et al., 2009]. Duplicates were marked using the `MarkDuplicates` tool from the Picard suite.

```
> 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")
```

The `design` matrix is specified such that it contains all the relevant experimental factors. In this case, only cell types are included but more complicated experiments can also be accommodated (e.g. batch effects, paired samples). Specification of the design matrix is explained in greater depth in the edgeR user's manual.

Chapter 2

Converting reads to counts

Hello there, traveller. This is a comment box that you'll see at the start of each chapter. It's meant to help you figure out which objects from previous chapters are needed to get the code in the current chapter to work. As we're starting out here, all we need are the `bam.files` we defined earlier.

The `windowCounts` function extracts read counts throughout the genome for a set of libraries using a sliding window approach. These counts can then be used to test for differential binding. Sorted and indexed BAM (i.e. binary SAM) files are required as input. A suitable fragment length should also be supplied for read extension, particularly for transcription factor datasets (see below).

```
> frag.len <- 110
> data <- windowCounts(bam.files, ext=frag.len)
> data$region
```

```
GRanges with 378858 ranges and 0 metadata columns:
           seqnames               ranges strand
              <Rle>            <IRanges>  <Rle>
       [1]    chr10   [3102701, 3102701]      *
       [2]    chr10   [3103851, 3103851]      *
       [3]    chr10   [3105651, 3105651]      *
       [4]    chr10   [3105701, 3105701]      *
       [5]    chr10   [3105751, 3105751]      *
       ...      ...                  ...    ...
  [378854]     chrY [90815051, 90815051]      *
  [378855]     chrY [90815151, 90815151]      *
```

```
  [378856]     chrY [90815201, 90815201]      *
  [378857]     chrY [90817401, 90817401]      *
  [378858]     chrY [90818451, 90818451]      *
  ---
  seqlengths:
                  chr10               chr11 ... chrY_JH584303_random
              130694993           122082543 ...                158099
```

Reads which have been marked as PCR duplicates can be ignored by setting `dedup=TRUE`. This can reduce the variability between replicates caused by inconsistent duplication. However, it also caps the number of reads at each position. This can lead to loss of power in high abundance regions and false positives when the same upper bound is applied to libraries of varying size. Thus, duplicate removal is generally not recommended for routine DB analyses.

Reads can also be filtered 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 may provide more reliable results. The exact value of the threshold depends on the range of scores provided by the aligner. For subread, a value of 100 is generally suitable.

```
> data2 <- windowCounts(bam.files, ext=frag.len, minq=100, dedup=TRUE)
> head(data2$counts)
```

```
     [,1] [,2] [,3] [,4]
[1,]    4    4    7    5
[2,]    9    9    7    6
[3,]    7    4    6    6
[4,]    6    5    4    5
[5,]    4    6    4    7
[6,]    3   10   13   11
```

2.1 Extracting count combinations

Each sequenced read in single-end data corresponds to one of the 5′ ends of a genomic DNA fragment. A rough estimate of the genomic interval spanned by the fragment corresponding to each read can be obtained by extending each sequenced read from its 5′ end in the direction of its 3′ end by the average fragment length. For TF experiments, a suitable value for the average fragment length can be estimated from the peak in a cross-correlation plot (see below). An alternative approach is to estimate the average fragment size from post-fragmentation gel electrophoresis images. Typical values range from 100 to 300 bp.

The number of extended reads overlapping a window in the genome corresponds to the number of DNA fragments enriched by immunoprecipitation (IP) at that window This is

a discrete count which is defined here as the "enrichment depth". With multiple libraries, each window will be associated with a count combination (one enrichment depth count per library). These count combinations are extracted across the genome and passed to the statistical routines inherited from edgeR. The aim is to determine whether there are significant differences in the enrichment depths between treatment groups.

2.1.1 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 count combinations are extracted for analysis. This may be useful when memory is limited on the machine. Of course, this sacrifices spatial resolution as adjacent positions are not counted and thus cannot be distinguished.

```
> data2 <- windowCounts(bam.files, spacing=100, ext=frag.len)
> data2$region
```

```
GRanges with 189164 ranges and 0 metadata columns:
           seqnames               ranges strand
              <Rle>            <IRanges>  <Rle>
       [1]    chr10   [3102701, 3102701]      *
       [2]    chr10   [3105701, 3105701]      *
       [3]    chr10   [3105801, 3105801]      *
       [4]    chr10   [3107301, 3107301]      *
       [5]    chr10   [3107601, 3107601]      *
       ...      ...                  ...    ...
  [189160]     chrY [90813601, 90813601]      *
  [189161]     chrY [90813801, 90813801]      *
  [189162]     chrY [90815001, 90815001]      *
  [189163]     chrY [90815201, 90815201]      *
  [189164]     chrY [90817401, 90817401]      *
  ---
  seqlengths:
                  chr10                chr11 ... chrY_JH584303_random
              130694993            122082543 ...                158099
```

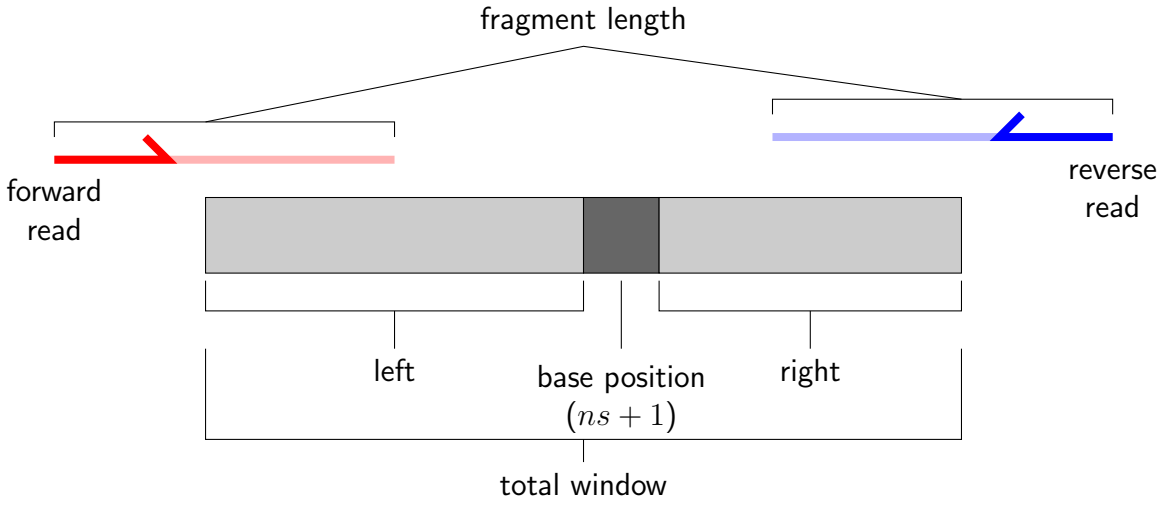
Windows with a low sum of counts across all libraries can be filtered out using the `filter` argument. This improves memory efficiency by discarding the majority of low-abundance windows corresponding to uninteresting background regions. The default filter value is set as the number of libraries multiplied by 5. This roughly corresponds to the minimum average count required for accurate downstream modelling. Note that more sophisticated filtering is recommended and can be applied later (see Chapter 4).

```
> data2 <- windowCounts(bam.files, ext=frag.len, filter=30)
> head(data2$counts)
```

```
     [,1] [,2] [,3] [,4]
[1,]   16   15   13   11
[2,]   16   11   10   13
[3,]   14    7    5   13
[4,]   13    5   10   14
[5,]   14    7    3   10
[6,]   12   17   12    5
```

2.1.2 Increasing the window size

The nominal "center" of each window is a single base position at $ns + 1$ for spacing s and some integer n. By default, the window has a width of 1 bp as it consists only of this base position. This is useful for analyses involving punctate regions of enrichment (e.g. transcription factors). For more diffuse marks, the window can be extended on either side by setting positive values for `left` and `right`.



Collection of reads across a wider region results in larger counts and greater power. The example below extends each window to the right by 999 bp to produce a 1 kbp window. However, spatial resolution is sacrificed when a large width value is used. Power can then be lost when reads from a non-DB region are counted along with a neighbouring DB loci. This is because the log-fold change is "diluted" by the addition of constant background.

```
> data2 <- windowCounts(bam.files, right=999, ext=frag.len, spacing=500)
> data2$region
```

```
GRanges with 4629172 ranges and 0 metadata columns:
            seqnames               ranges strand
               <Rle>            <IRanges>  <Rle>
        [1]    chr10   [3101001, 3102000]      *
        [2]    chr10   [3101501, 3102500]      *
        [3]    chr10   [3102001, 3103000]      *
        [4]    chr10   [3102501, 3103500]      *
        [5]    chr10   [3103001, 3104000]      *
        ...      ...                  ...    ...
  [4629168]     chrY [90838501, 90839500]      *
  [4629169]     chrY [90839001, 90840000]      *
  [4629170]     chrY [90839501, 90840500]      *
  [4629171]     chrY [90842501, 90843500]      *
  [4629172]     chrY [90843001, 90844000]      *
  ---
  seqlengths:
                 chr10                chr11 ... chrY_JH584303_random
             130694993            122082543 ...                158099
```

The window size can be interpreted as a measure of the width of the binding site. Thus, TF analyses will usually use a small window size. 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 width values can be combined to provide a comprehensive picture of differential binding at all resolutions.

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 width.

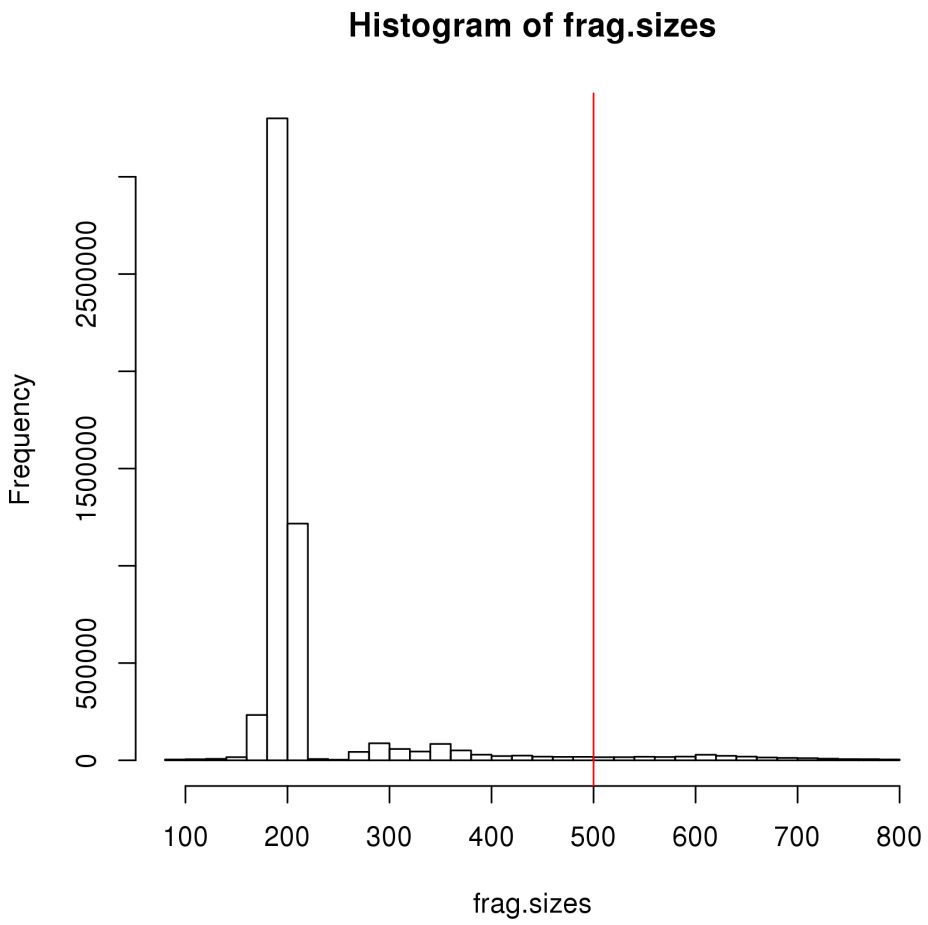
2.2 Experiments involving paired-end tags

ChIP experiments with paired-end tag (PET) sequencing can be accomodated by setting `pet=TRUE` in the `windowCounts` function call. Read extension is not required for ChIP-PET data as the genomic interval spanned by the originating fragment is explicitly defined by the positions of the paired reads. The maximum allowable size of any fragment is set using the `max.frag` argument.

```
> petBamFile <- "example-pet.bam"
> max.len <- 500
> data.pet <- windowCounts(petBamFile, max.frag=max.len, pet=TRUE)
```

The distribution of fragment sizes can be extracted using the `getPETSizes` function. This can be plotted and used to define the maximum allowable fragment length. A suitable value for this example would be about 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)
> abline(v=max.len, col="red")
```



Valid read pairs consist of two reads where both 3′ ends lie in the genomic interval between the 5′ ends. The size of the interval must also be between the read length and the specified

maximum. Invalid pairs and singleton reads are ignored during processing. The proportion of such reads can be determined by counting the total number of reads and subtracting twice the number of valid pairs. A non-negligble proportion of invalid reads indicates that there are problems with PET sequencing.

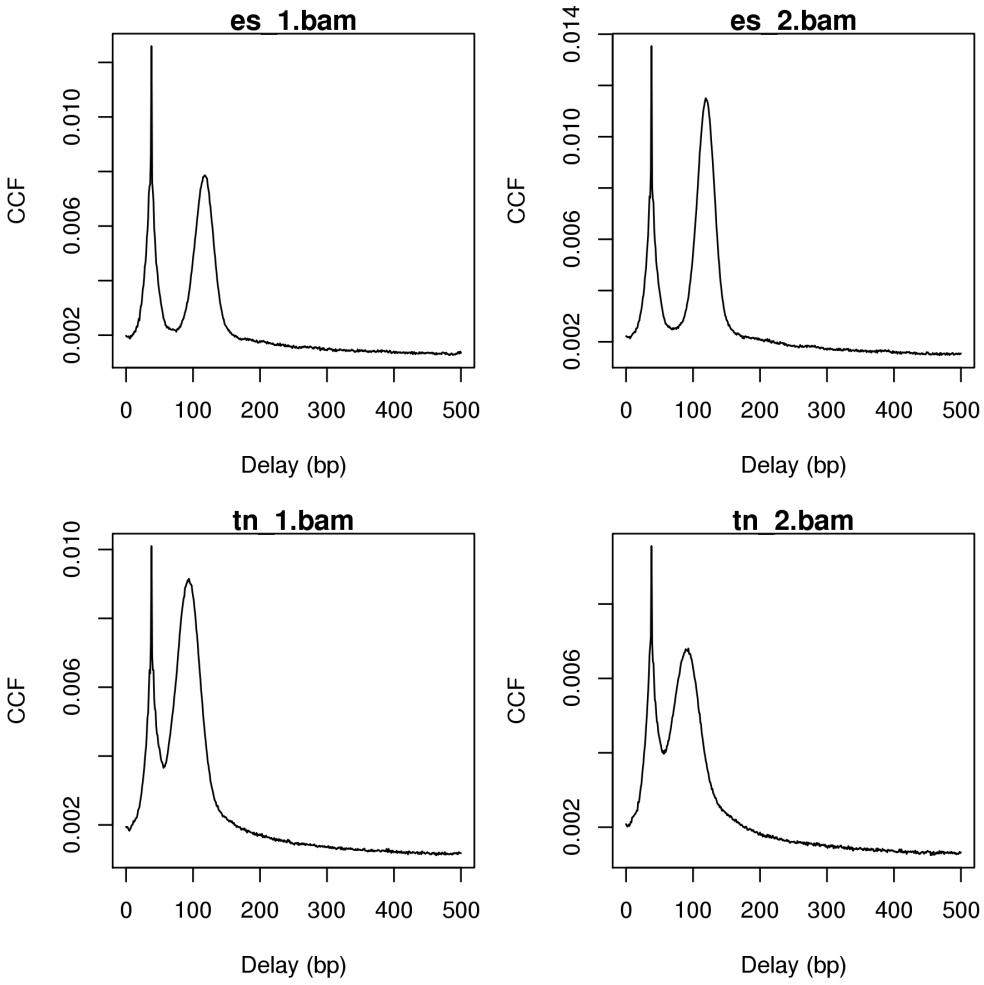
```
> out$diagnostics

        total        single mate.unmapped     inter.chr
     12717430             0             5         15181

> 1-sum(out$sizes < max.len)*2/out$diagnostics[1]

    total
0.1677828
```

Note that the PET-based algorithms in csaw depend on the synchronisation of mate information for each alignment in the BAM file. Any file manipulations which might break this synchronisation should be followed by a call to the FixMateInformation program from the Picard suite.

2.3 Checking immunoprecipitation efficiency

Cross-correlation plots can be generated directly from BAM files using the `correlateReads` function. These plots provide a measure of the IP efficiency of a TF ChIP-seq experiment [Kharchenko et al., 2008]. The correlations between read positions on the forward and reverse strands are plotted as a function of the lag distance. When the pulldown is successful, the correlations should peak smoothly at a distance corresponding to the average fragment length. This reflects the strand-dependent bimodality of reads around TF binding sites. The location of the peak thus provides an estimate for the average fragment length ($\sim$ 110 bp below) for use in read extension.

```
> max_distance <- 500
> par(mfrow=c(2,2), mar=c(5, 4, 1, 1))
> for (b in bam.files) {
+   x <- correlateReads(b, max_distance, dedup=TRUE, cross=TRUE)
+   plot(0:max_distance, x, type="l", ylab="CCF", xlab="Delay (bp)", main=b)
+ }
```

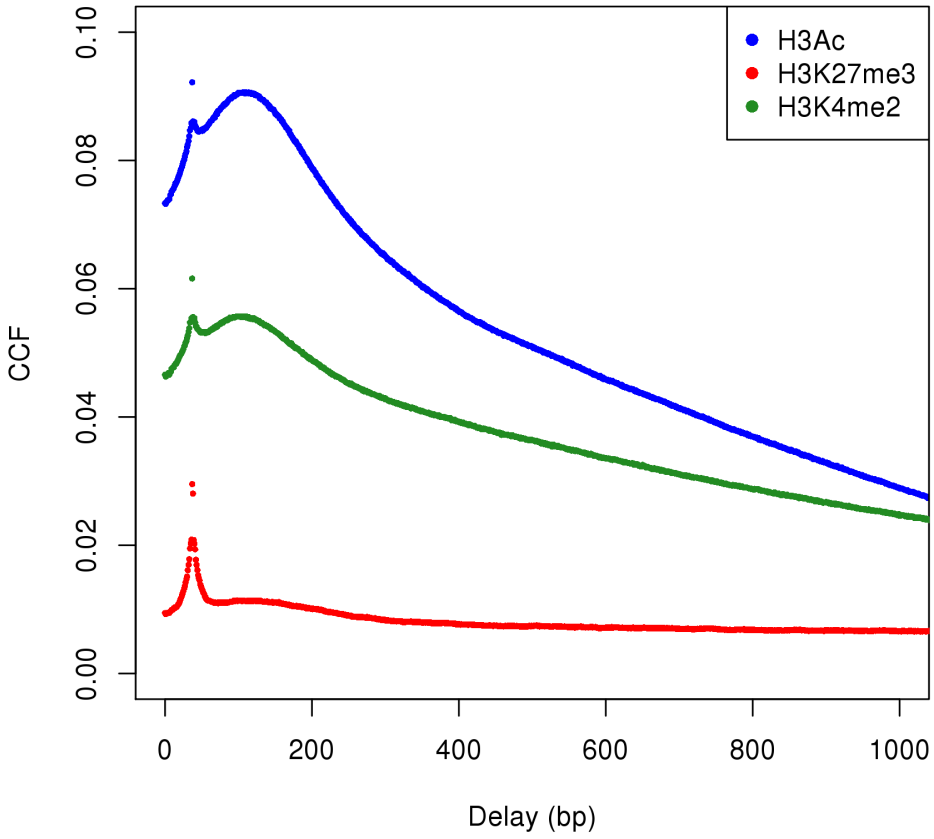


A sharp spike may also observed in the plot at a distance corresponding to the read length. This is an artefact which is thought to be a result of the preference of aligners for uniquely mapped reads. The size of the smooth peak can be compared to the height of the spike to assess the signal-to-noise 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.

These plots can also be used for fragment length estimation of punctate histone marks such as histone acetylation and H3K4 methylation. However, they are less effective for 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, cex=0.5)
> points(0:n, h3k27me3, col="red", pch=16, cex=0.5)
> points(0:n, h3k4me2, col="forestgreen", pch=16, cex=0.5)
> legend("topright", col=c("blue", "red", "forestgreen"),
+   c("H3Ac", "H3K27me3", "H3K4me2"), pch=16)
```



2.4 Ensuring synchronisation

In practice, most ChIP-seq analysis pipelines based on csaw will involve multiple `windowCounts` calls. Users should supply the same values for the `bam.files`, `dedup`, `pet`, `minq` and (for paired-end data) `max.frag` 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)
> data2 <- countWindows(param, ext=frag.len, filter=50)
> data3 <- countWindows(param, ext=frag.len, right=1000)
> data4 <- countWindows(param, bin=10000)
```

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. That said, for the sake of simplicity, the examples in the rest of this guide will use the `windowCounts` function. This is equivalent to read counting with the default values for each of the critical parameters previously described.

Chapter 3

Calculating normalization factors

Still there? Good. This next chapter will need the `bam.files` vector again. If you bother to keep reading, you'll notice that a number of other files also get called. However, these are mostly for demonstration purposes and aren't necessary for the main example.

Highly enriched regions consume more sequencing resources and suppress the representation of background regions in the sequencing output. Uneven undersampling of the background between libraries can result in false positives whereby the background regions in one library are "enriched" compared to the undersampled background regions in another library. Total library scaling fails to correct for undersampling as it does not make any distinction between background and enriched regions. Rather, normalization procedures such as the TMM method [Robinson and Oshlack, 2010] are recommended to equalize the representation of background regions between libraries.

Unforunately, counts from background regions in ChIP-seq libraries are usually too low for fold change-based normalization procedures. Zero counts will produce undefined M-values which are difficult to interpret. Such values are usually discarded which can lead to considerable loss of information when many zeros are present. Adding a prior count to avoid undefined values 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.

3.1 Binning for higher counts

The problems associated with low counts can be avoided with a binning approach. Reads are counted into contiguous non-overlapping bins across the genome. This results in larger

counts as reads are collected over a wider region. The binned counts are then passed to the TMM algorithm for calculation of the normalization factors. Briefly, M-values (i.e. library size-adjusted log-expression ratios) are calculated across all bins for each pair of libraries. Bins with extreme M-values are trimmed away. A scaling factor is then calculated as the mean M-value across the remaining bins.

```
> binned <- windowCounts(bam.files, bin=10000)
> normfacs <- normalizeChIP(binned$counts)
> normfacs

[1] 0.9843100 0.9594416 1.0334848 1.0245790
```

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. A value around 10000 bp is usually suitable. Testing multiple bin sizes is recommended to ensure that the estimates are robust to any changes.

```
> binned <- windowCounts(bam.files, bin=5000)
> normalizeChIP(binned$counts)

[1] 0.9840325 0.9611444 1.0318386 1.0246845

> binned <- windowCounts(bam.files, bin=15000)
> normalizeChIP(binned$counts)

[1] 0.9848029 0.9578794 1.0344662 1.0247632
```

Normalization factor estimates should also be robust to the addition of a range of prior counts. Substantial differences upon addition of a prior or changes to the bin size suggest that the counts are too low for stable estimation. Switching to a larger bin size is recommended to obtain larger counts. Of course, the choice of bin size also depends on the sequencing depth. Deeper sequencing means that smaller bins can be used without running into problems with low counts.

```
> mult <- binned$total/exp(mean(log(binned$total)))
> normalizeChIP(t(t(binned$counts)+0.1*mult), binned$total)

[1] 0.9853443 0.9584285 1.0339557 1.0241188

> normalizeChIP(t(t(binned$counts)+1*mult), binned$total)

[1] 0.9855105 0.9586935 1.0337155 1.0239010
```

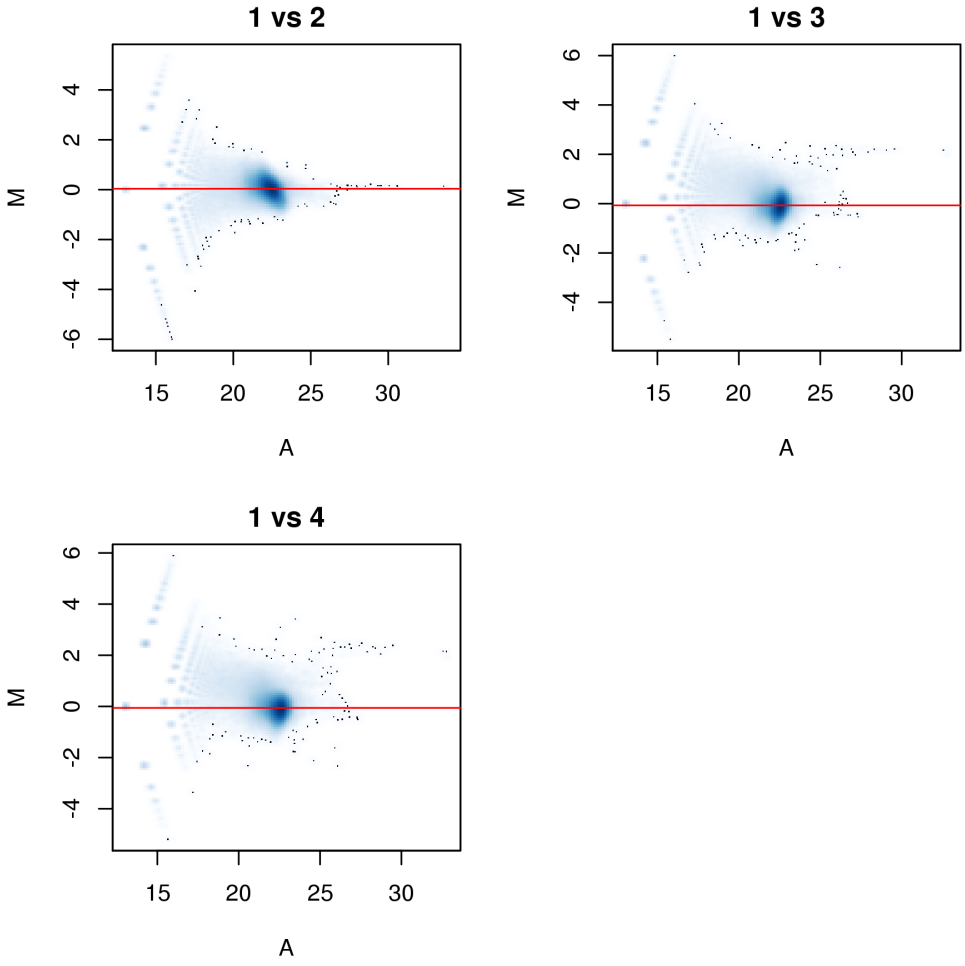
```
> normalizeChIP(t(t(binned$counts)+5*mult), binned$total)

[1] 0.9864867 0.9601693 1.0320971 1.0229170
```

It's worth noting that `normalizeChIP` does not perform precision weighting by default. Weighting in the original TMM method is designed to increase the contribution of points with high counts which should be more precise. However, differential binding is more likely to occur in high-abundance bins corresponding to enriched regions. If any DB regions should survive trimming (e.g. those with subtle fold changes), upweighting them during calculation of the global scaling factor will be counterproductive.

3.2 Visualizing the normalization factors

The fold-change distributions of the bins can be visualised using a MA plot. Most plots should contain a single dark circular mass. This represents the background regions making up most of the genome. Separation into multiple discrete points indicates that the counts are too low and that larger bin sizes should be used. Undersampling then manifests as a shift in the position of this mass on the vertical axis. Ideally, the logarithm of the ratios of the corresponding normalization factors should correspond to the centre of this mass. This indicates that undersampling has been corrected.

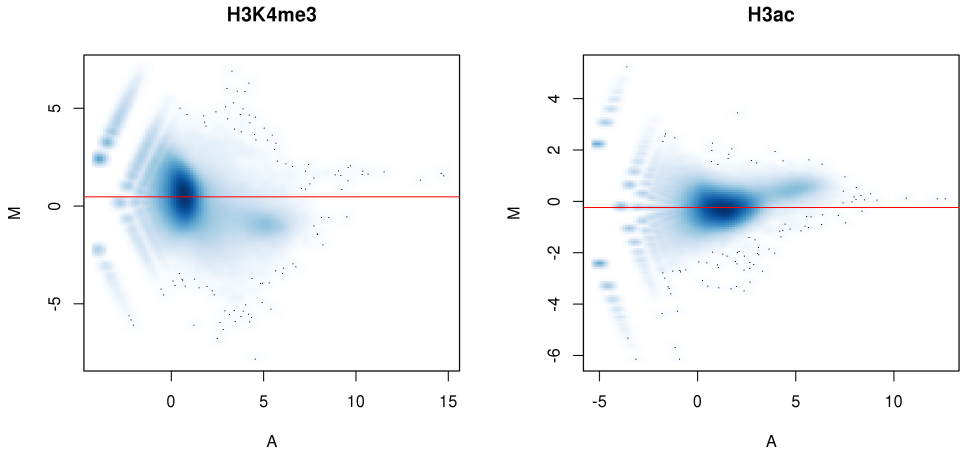
```
> num <- length(bam.files)-1;
> root <- round(sqrt(num));
> par(mfrow=c(root, ceiling(num/root)), mar=c(5, 4, 2, 1.5));
> adj.counts <- cpm(binned$counts, log=TRUE)
> for (i in 1:num) {
+   cur.x <- adj.counts[,1]
+   cur.y <- adj.counts[,1+i]
+   smoothScatter(x=(cur.x+cur.y)/2+6*log2(10), y=cur.x-cur.y,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



Some MA plots may have a mass of points at high A-values which diverges from the normalization line. This usually represents regions which are differentially bound between groups. However, differential binding can also be observed between replicates. This is a batch effect caused by differences in IP efficiency between libraries. An abundance trend among the mass of background regions is also consistent with a batch effect from general differences in library preparation.

```
> comparisons <- list(H3K4me3=c("h3k4me3_mat.bam", "h3k4me3_pro.bam"),
+   H3ac=c("h3ac.bam", "h3ac_2.bam"))
> par(mfrow=c(1,2))
> for (x in names(comparisons)) {
+   binned <- windowCounts(comparisons[[x]], bin=10000L)
+   normfac <- normalizeChIP(binned$count)
+   adj.counts <- cpm(binned$counts, log=TRUE)
+   cur.x <- adj.counts[,1]
```

```
+   cur.y <- adj.counts[,2]
+   smoothScatter(x=(cur.x+cur.y)/2, y=cur.x-cur.y,
+     xlab="A", ylab="M", main=x)
+   abline(h=log2(normfac[1]/normfac[2]), col="red")
+ }
```



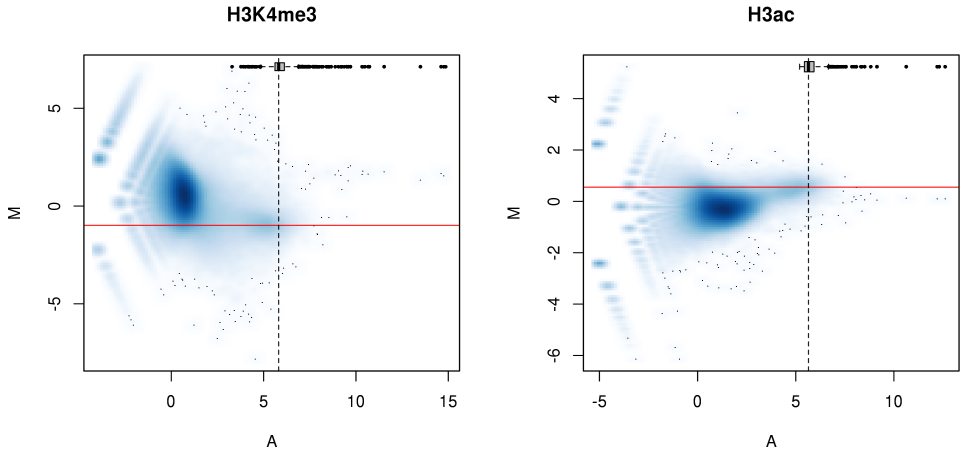
3.3 Normalizing against batch effects

The TMM-based normalization procedure described previously will attempt to equalize the level of background enrichment. This will amplify any global differences in enrichment between libraries from different groups. However, it can also increase the size of any IP-driven batch effects between replicates. The latter effect will inflate the observed variability which will usually offset any increases in power from the former. Thus, alternative normalization approaches are required to deal with strong batch effects.

The simplest method assumes that most enriched regions are not DB. The TMM method can then be applied exclusively to high abundance bins to correct for any IP-driven differences between replicates. Note that this also requires some definition of enriched regions. Here, the top 1% of high-abundance bins are assumed to be enriched (and, therefore, mostly constant between libraries). MA plots can be used to ensure that the normalization factor passes through the cloud of enriched bins. The boxplot describes the range of A-values corresponding to enriched bins.

```
> par(mfrow=c(1,2))
> threshold <- 0.01
> for (x in names(comparisons)) {
+   binned <- windowCounts(comparisons[[x]], bin=10000L)
```

```
+   ab <- aveLogCPM(binned$counts)
+   keep <- rank(ab)>(1-threshold)*length(ab)
+   normfac <- normalizeChIP(binned$count[keep,], lib.sizes=binned$totals)
+   adj.counts <- cpm(binned$counts, log=TRUE)
+
+   a <- (adj.counts[,1]+adj.counts[,2])/2
+   m <- adj.counts[,1]-adj.counts[,2]
+   smoothScatter(x=a, y=m, xlab="A", ylab="M", main=x)
+   abline(h=log2(normfac[1]/normfac[2]), col="red")
+   boxplot(a[keep], horizontal=TRUE, at=max(m), add=TRUE,
+     col="grey", pch=16, cex=0.5)
+   abline(v=median(a[keep]), lty=2)
+ }
```



These normalization factors can reduce the observed variance by removing any systematic differences between libraries. However, genuine biological differences may also be removed when overall enrichment is truly lower for one condition. For such analyses, the assumption of a non-DB majority amongst high-abundance bins is inappropriate. The choice of threshold may also be arbitrary if there is no obvious demarcation between background and enriched bins. Finally, some filtering is required to ensure that the rest of the analysis is consistent with the assumed proportion of enriched regions (see Chapter 4).

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

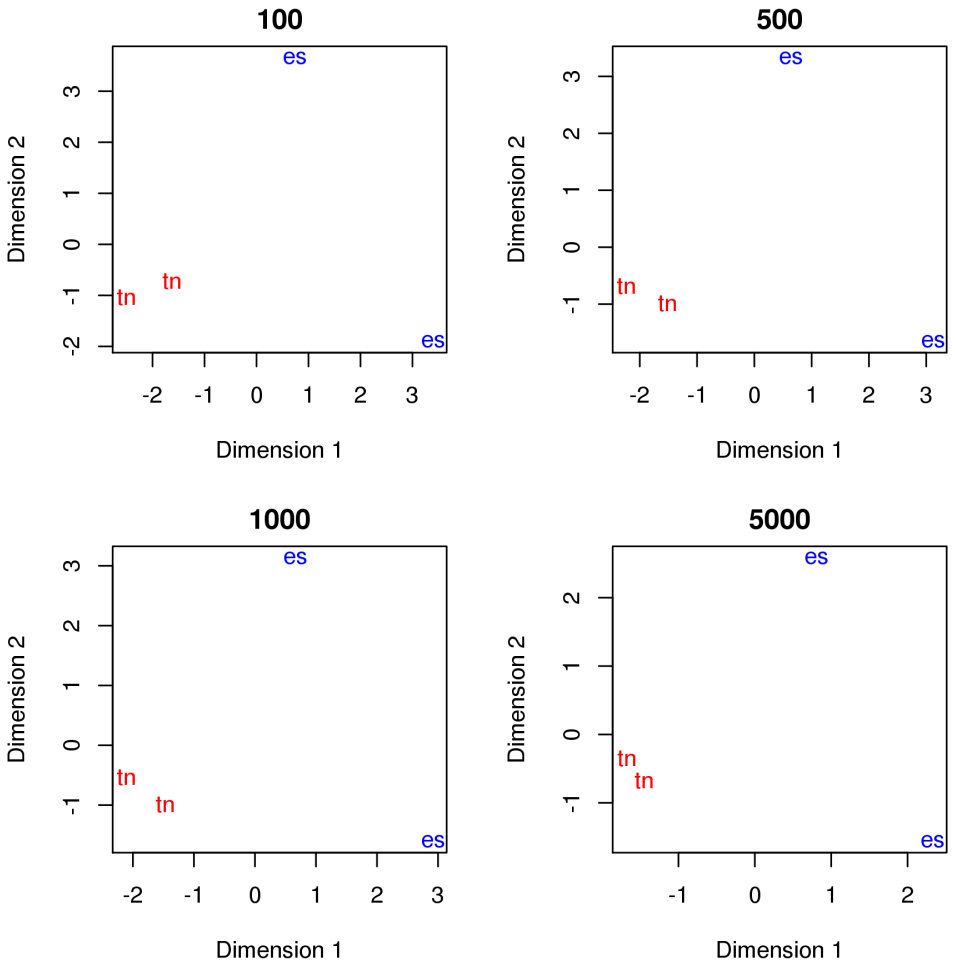
between libraries. This is generally reasonable as the biases depend on fixed properties of the genome sequence such as GC content. Thus, biases should cancel out during DB comparisons. Any variability in the bias between samples will be absorbed into the dispersion estimate. For example, sample-specific biases related to GC content [Risso et al., 2011, Cheung et al., 2011] are effectively treated as GC-correlated variances in csaw.

That said, explicit normalization to correct these biases can improve results for some datasets. This is due to decreases in the observed variability when systematic differences between libraries are removed. 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

On a 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 mean (square root of the squared) log-fold change across the top set of bins with the highest absolute log-fold changes. A plot using a small `top` set visualizes the most extreme differences whereas a plot using a large set visualizes overall differences. The use of binned counts is necessary as the MDS plot is based on fold-changes between libraries.

```
> par(mfrow=c(2,2), mar=c(5,4,2,2))
> binned <- windowCounts(bam.files, bin=2000)
> 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 reproducibility is high and that the effect sizes are large. Mixing between replicates of different conditions indicates that the biological difference has no effect on protein binding. An alternative interpretation is that the variability in the data is too large for any effect to be observed. 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

Ding ding! This is the halfway mark - or, at least it was when this thing was written. If you're seeing something different, it means that - wait for it - you're in the *future*. I'd come and introduce myself in person but my flux capacitor is broken. Anyway, back to you. This part will require the `data` list generated from Chapter 2 and the `normfacs` vector from Chapter 3.

Many of the low abundance windows in the genome correspond to background regions in which differential binding is not expected. In addition, windows with low count sums will have large p-values even if all the counts were associated with one treatment condition. Removing such uninteresting tests reduces the severity of the multiple testing correction and increases detection power amongst the remaining tests. Filtering is valid so long as it is independent of the test statistic under the null hypothesis [Bourgon et al., 2010]. In the NB framework, this corresponds to filtering on the overall mean (based on an analogy to the normal case). Row sums can also be used for datasets where effective library sizes are not hugely different.

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -2.110  -2.025  -1.877  -1.628  -1.485  12.460
```

For demonstration purposes, an arbitrary threshold of -1 is used here to filter the window abundances. Users can then restrict the dataset to the filtered values (though in this case, a copy is used so as to not affect downstream demonstrations). While filtering can be performed at any stage of the analysis, 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 successful information sharing (see Chapter 5).

```
> keep <- abundances > -1
> data2 <- data
> data2$counts <- data2$counts[keep,]
> data2$region <- data2$region[keep]
> sum(keep)
```

```
[1] 46356
```

The exact choice of filter threshold may not be immediately obvious. A filter which is too conservative will be ineffective whereas a filter which 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.

4.1 By proportion

One approach is to to assume that only a certain proportion - say, 0.1% - of the genome is genuinely enriched. Only the top proportion of high-abundance windows are then retained. 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 total amount of enrichment can vary.

```
> spacing <- 50L
> desired <- 0.001
> max.windows <- sum(floor(seqlengths(data$region)/spacing))
> keep <- rank(abundances) > max.windows*(1-desired)
> sum(keep)
```

```
[1] 0
```

4.2 By global enrichment

An alternative approach involves choosing a filter based on enrichment over 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=bin.size)
> y.bin <- DGEList(binned$counts, norm.factors=normfacs)
> bin.ab <- aveLogCPM(y.bin)
> threshold <- median(bin.ab)
```

Note that the threshold needs to be adjusted to allow direct comparisons to window abundances. This is done by assuming that reads are uniformly distributed within each window/bin in background regions. The effective size of a window can then be defined as the sum of the read extension length and the window size. In other words, a hypothetical bin with width equal to the effective window size should have the same expected read count as that window. The adjustment can then be computed from the ratio of the bin size to the effective window size. Simply using the window size directly is not sufficient due to directional read extension.

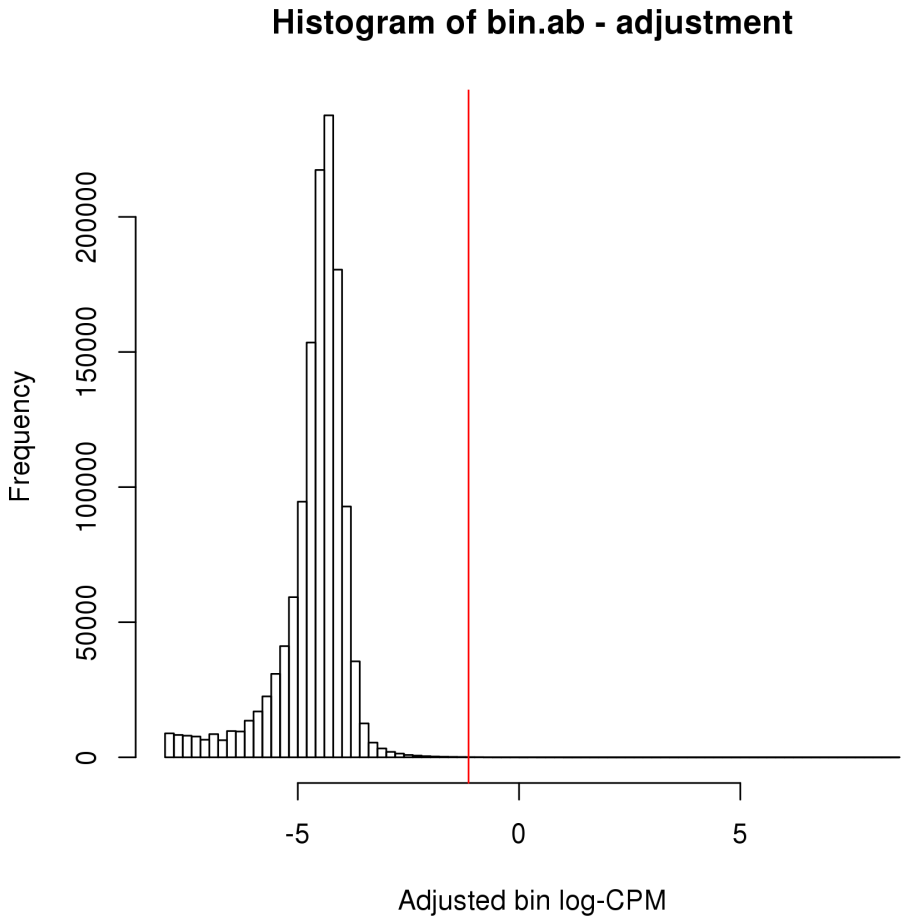
```
> width <- median(width(data$region))
> eff.win.size <- width+frag.len
> adjustment <- log2(bin.size/eff.win.size)
> threshold <- threshold-adjustment
```

Windows are then filtered based on some minimum required fold-change over the global background. Here, a fold change of 10 over the background coverage is necessary for a window to be considered as enriched. This approach has an intuitive and experimentally relevant interpretation which adapts to the level of non-specific enrichment in the dataset. However, it may be too aggressive when genuine enrichment is weak.

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

[1] 56901
```

The effect of filtering can also be visualized with a histogram. This allows users to confirm that the bulk of (assumed) background regions are discarded upon filtering. Note that regions of true enrichment will usually fail to appear on such plots due to the dominance of low-abundance regions across the genome.

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



4.3 By local enrichment

Local background estimators can also be constructed with a little imagination. This avoids inappropriate filtering due to differences in the sequencing coverage across the genome. Here, the average abundance of the 2 kbp region surrounding each window will be used as a local estimate of non-specific enrichment for that window. Note that read count loading uses both `left` and `right` parameters as regions on both sides of the window are desired.

```
> wider <- windowCounts(bam.files, left=bin.size/2L, right=bin.size/2L-1L)
```

Each window can be matched to the corresponding bin representing the "neighbourhood" of that region. Counts for each window are then subtracted from the counts for its neighbourhood. This ensures that enriched regions inside the window do not interfere with estimation of the local background.

```
> suppressWarnings(expanded <- trim(resize(data$region, bin.size, fix="center")))
> matched <- findOverlaps(expanded, wider$region, type="equal", select="first")
> neighbour.counts <- wider$counts[matched,]-data$counts
```

Some adjustment for width is again required for proper comparisons between the abundances of each window and its neighbourhood. In this case, the adjustment must also account for the subtraction of window counts from those of the neighbourhood interval.

```
> adjustment <- log2((bin.size-eff.win.size)/eff.win.size)
```

Enrichment is then 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.

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -9.486   1.389   1.937   1.864   2.365  11.060
```

Note that this procedure also assumes that no other enriched regions are present in each neighbourhood. Otherwise, the local background will be overestimated. Windows containing enriched regions may then 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.

4.4 By prior information

When only a subset of genomic regions are of interest, power can be improved by removing windows lying outside of these regions. Annotated features of interest include promoters and enhancers (for local and distal transcriptional regulation) as well as exons (for alternative promoters or splicing). On the flip side, repeat-masked regions are commonly removed to avoid artifactual differences from copy number variations. The example below retrieves the coordinates of the broad gene bodies from the mouse genome after extending each 5′ end by 3 kbp.

```
> require(org.Mm.eg.db)
> start.anno <- toTable(org.Mm.egCHRLOC)
> ends <- toTable(org.Mm.egCHRLOCEND)[,2]
```

```
> extension <- 3000
> coord5 <- ifelse(ends>0, start.anno[,2]-extension, -ends)
> broads <- GRanges(seqnames=paste0("chr", start.anno[,3]),
+  IRanges(coord5, coord5+extension))
> broads

GRanges with 27295 ranges and 0 metadata columns:
          seqnames                ranges strand
             <Rle>             <IRanges>  <Rle>
      [1]     chr6 [128526720, 128529720]      *
      [2]    chr11 [116590687, 116593687]      *
      [3]    chr11 [120047145, 120050145]      *
      [4]    chr11 [120041781, 120044781]      *
      [5]     chr4 [ 53159895,  53162895]      *
      ...      ...                 ...    ...
  [27291]     chrY   [ 3306449,  3309449]      *
  [27292]     chrY   [ 2827680,  2830680]      *
  [27293]     chrY   [ 2897989,  2900989]      *
  [27294]     chrY   [ 3768673,  3771673]      *
  [27295]    chr14   [46575851, 46578851]      *
  ---
  seqlengths:
                   chr1                chr10 ...                 chrY
                     NA                   NA ...                   NA
```

Windows can then 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. Note that filtering in this manner is complementary to abundance filtering. In fact, the two can be combined for greatest effect.

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

[1] 36120
```

It should be stressed that any information used here should be independent of the DB status under the null hypothesis in this 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.

Chapter 5

Testing differential binding

Breaker, breaker... oh, you're still awake. Fine, let's keep going. This part will require the `data` list produced in Chapter 2 and filtered in Chapter 4. You'll also - what's that? It wasn't properly filtered because I put the result in a dummy variable? Well, here you go:

```
> original <- data
> data <- data2
```

Are you happy? Don't answer that, I don't actually care. Now, you'll need the `normfacs` vector from Chapter 3. You'll also want to dig out the `design` matrix from the introduction.

Low counts per window are typically observed in ChIP-seq datasets, even for regions of relative enrichment. The statistical analysis must be able to handle discreteness in the data so software packages using count-based models are recommended. In this guide, the quasi-likelihood (QL) framework in the edgeR package is used [Lund et al., 2012]. Each window count can then be directly modelled by the negative binomial (NB) distribution to account for variability between biological replicates [Robinson and Smyth, 2008].

Before proceeding with the analysis, a `DGEList` object must be formed from the count matrix. Additional information like the library size and the normalization factors can (and should) be added. For this analysis, the `normfacs` vector from TMM normalization of background bins is used. However, alternative values can also be supplied. If an offset matrix is necessary (e.g. quantile normalization), this can be supplied downstream by using the `offset` argument in most edgeR functions.

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

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 then 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.

```
> 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"
```

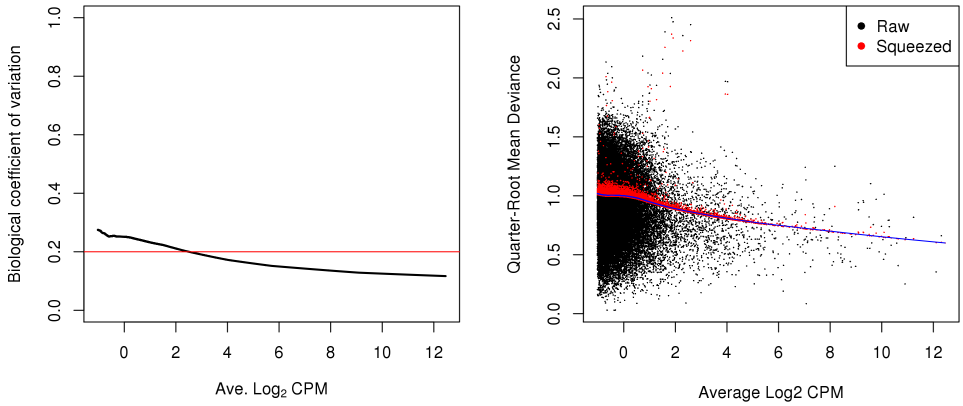
5.1 Estimating the dispersions

Under the QL framework, both the QL and NB dispersions are used to model biological variability in the data. The former ensures that the NB mean-variance relationship is properly specified with appropriate contributions from 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 value.

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 using the trended NB dispersion. A second mean-dispersion trend is fitted to the 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 computed from the heteroskedasticity of the data.

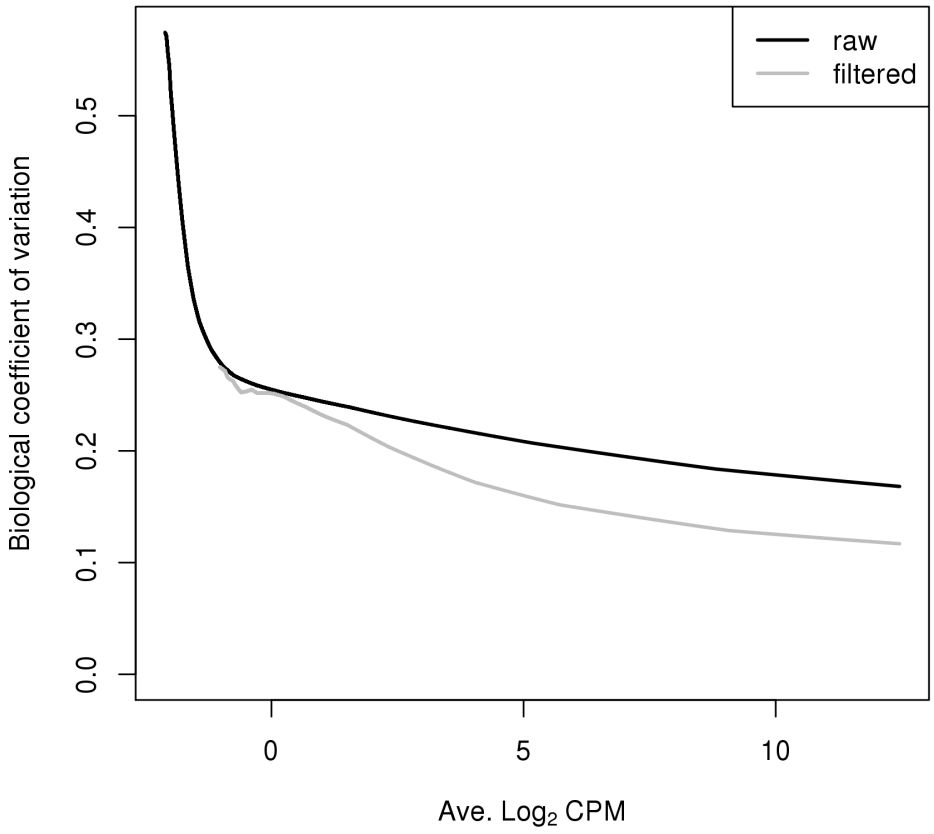
```
> par(mfrow=c(1,2))
> y <- estimateDisp(y, design)
> o <- order(y$AveLogCPM)
> plot(y$AveLogCPM[o], sqrt(y$trended.dispersion[o]), type="l", lwd=2,
+   ylim=c(0, 1), xlab=expression("Ave."~Log[2]~"CPM"),
+   ylab=("Biological coefficient of variation"))
> abline(h=0.2, col="red")
> results <- glmQLFTest(y,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

The effect of EB stabilisation can then 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 irregulaties may be indicative of an underlying structure in the data which cannot be modelled with the mean-dispersion trend. Non-random patterns in the raw dispersions may also be observed. This suggests that more aggressive filtering is required to remove low counts which cannot be handled with asymptotic GLM theory.



A strong trend may 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. However, it is difficult for empirical methods to accurately capture strong trends. This results in increased errors in the fitted values, typically manifesting as overestimation of dispersions at high abundances. Users should check whether removal of the low abundance regions affects the dispersion estimate. Large differences imply that further filtering may be desirable (see 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)
```



5.1.1 Modelling heteroskedasticity

The heteroskedasticity of the data is modelled in edgeR by the prior degrees of freedom. A large prior degrees of freedom represents strong shrinkage and low heteroskedasticity. This is because a greater amount of EB shrinkage can be performed to reduce uncertainty and maximize power when the dispersions are not variable. However, strong shrinkage is not appropriate if the dispersions are highly variable. Lower 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.
  0.524  54.580  54.580  53.720  54.580  54.580
```

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 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. Alternative normalization methods may be necessary to resolve this issue (Section 3.3).

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 power.

5.2 Testing for DB windows

The effect of specific factors can then 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.

```
> head(results$table)
```

```
       logFC     logCPM           F    PValue
1 -0.5890404 -0.9244610 1.129627061 0.2923701
2  0.2149303 -0.9900990 0.141287224 0.7084103
3 -0.4017381 -0.8226146 0.546830889 0.4626725
4 -0.5489246 -0.2593132 1.359337172 0.2485435
5 -0.5158578 -0.8604284 0.919915616 0.3415777
6  0.0295069 -0.1631215 0.004036486 0.9495657
```

More formally, the null hypothesis is that a specified factor in the design matrix has no effect i.e. the mean enrichment depth is equal across treatment conditions. The `contrast` parameter 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` so changes between cell types can be identified. Specification of the contrast is explained in greater depth in the edgeR user's manual.

Chapter 6

Correction for multiple testing

All right, we're almost there. This chapter needs the `results` list and the `y` object (actually a `DGEList`, but never mind) from the last chapter. You'll also need the `data` list and the `frag.len` value from Chapter 2, as well as the the `normfacs` vector from Chapter 3.

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. The most obvious approach is to apply the BH method to the set of p-values across all windows. However, the FDR across all detected windows which is not necessarily the error of interest. Interpretation and validation of ChIP-seq experiments are more concerned with regions of the genome. Such regions will usually consist of one or more overlapping windows.

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. The first set of windows represents a truly DB event 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, validation will only be performed once on each cluster. This cluster-based interpretation results in an actual FDR of 50% and reduced efficiency of validation.

6.1 Controlling the cluster FDR

Problems from misinterpretation can be avoided by applying the BH method to p-values from each cluster of windows. Windows can be clustered together using a number of strategies (see below). 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 converted to q-values using the BH method to control the cluster FDR.

6.1.1 Clustering with external information

Combined p-values can be computed for a predefined set of regions based on the windows inside each region. The most obvious source of predefined regions is that of annotated features such as promoters or gene bodies. Alternatively, peak-calling programs such as MACS [Zhang et al., 2008] can be used to define peaks for clustering. However, some care is required to avoid data snooping. One approach is to pool all libraries into a single file and use it to call peaks in single-sample mode. In either case, the `findOverlaps` machinery from the GenomicRanges package can be used to identify all windows in or overlapping each specified region.

```
> promoter <- GRanges(c("chr10", "chr3"),
+   IRanges(c(3984701, 27152751), c(3987701, 27155751)))
> olap <- findOverlaps(promoter, data$region)
> olap
```

```
Hits of length 7
queryLength: 2
subjectLength: 46356
  queryHits subjectHits
   <integer>   <integer>
 1         1         230
 2         1         231
 3         1         232
 4         2       31327
 5         2       31328
 6         2       31329
 7         2       31330
```

The `combineFDR` function can then be used to combine the p-values for all windows in each region. This provides a single combined p-value (and the corresponding q-value) for each region. Note that the row names of the output table correspond to the value of the integer identifiers supplied in `ids`. These should, in turn, correspond to the regions of interest in `promoter`. The average log-CPM and log-FC for all windows in each region are also computed.

```
> ids <- queryHits(olap)
> wids <- subjectHits(olap)
> tabprom <- combineFDR(ids, results$table[wids,])
> head(tabprom)
```

```
       logFC     logCPM       PValue          FDR
1 0.05290579 -0.6904882 4.048857e-01 4.048857e-01
2 2.98715298 -0.5528216 9.358006e-08 1.871601e-07
```

At this point, one might imagine that a simpler solution to computing a cluster p-value would be to just collect and analyze counts over predefined peaks or promoter regions of interest. This is a valid strategy but provides poor performance for complex DB events. Consider a promoter region containing two separate peaks which are identically DB in opposite directions. Counting reads across the promoter will give equal counts for each group. Changes within the promoter will not be detected. Similar issues can be encountered for peak-based methods when the boundaries of each called peak are ambiguous. Window-based methods are more robust as each interval of the peak or region is examined separately. Neighbouring DB and non-DB regions can then be distinguished with greater success.

6.1.2 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 decision as to the threshold distance for separate binding events.

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

```
GRanges with 12548 ranges and 0 metadata columns:
          seqnames               ranges strand
             <Rle>            <IRanges>  <Rle>
      [1]    chr10   [3105651, 3105651]      *
      [2]    chr10   [3131101, 3131101]      *
      [3]    chr10   [3132751, 3132801]      *
      [4]    chr10   [3133901, 3136351]      *
      [5]    chr10   [3145051, 3146751]      *
      ...      ...                  ...    ...
  [12544]     chrY [90782851, 90782901]      *
  [12545]     chrY [90784051, 90784101]      *
  [12546]     chrY [90805151, 90805201]      *
  [12547]     chrY [90808851, 90808851]      *
  [12548]     chrY [90812101, 90813551]      *
  ---
  seqlengths:
```

```
                chr10                 chr11 ... chrY_JH584303_random
            130694993             122082543 ...                158099
```

Combined p-values are computed for each cluster as previously described. Application of the BH method then controls the FDR across all detected clusters. Like before, the row names of the output table point towards the corresponding coordinates of the clusters in `merged$regions`.

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

       logFC     logCPM     PValue       FDR
1 -0.5890404 -0.9244610 0.29237007 0.5407015
2  0.2149303 -0.9900990 0.70841030 0.8789808
3 -0.4753313 -0.5409639 0.46267247 0.7041912
4 -0.1639262 -0.4006806 0.58627888 0.8029499
5 -0.2635541 -0.6345128 0.92270284 0.9785287
6 -1.1254360 -0.9764751 0.05801115 0.1876753
```

If many overlapping windows are present, very large clusters may be formed which 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 is required to remove background regions. This mitigates chaining effects by increasing the average spacing between each window. Note that several large clusters may still be present due to high coverage within long tandem repeat loci.

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     1.0    51.0   250.3   101.0 51400.0
```

More generally, this problem can be avoided 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.

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1      51     245     151    4951
```

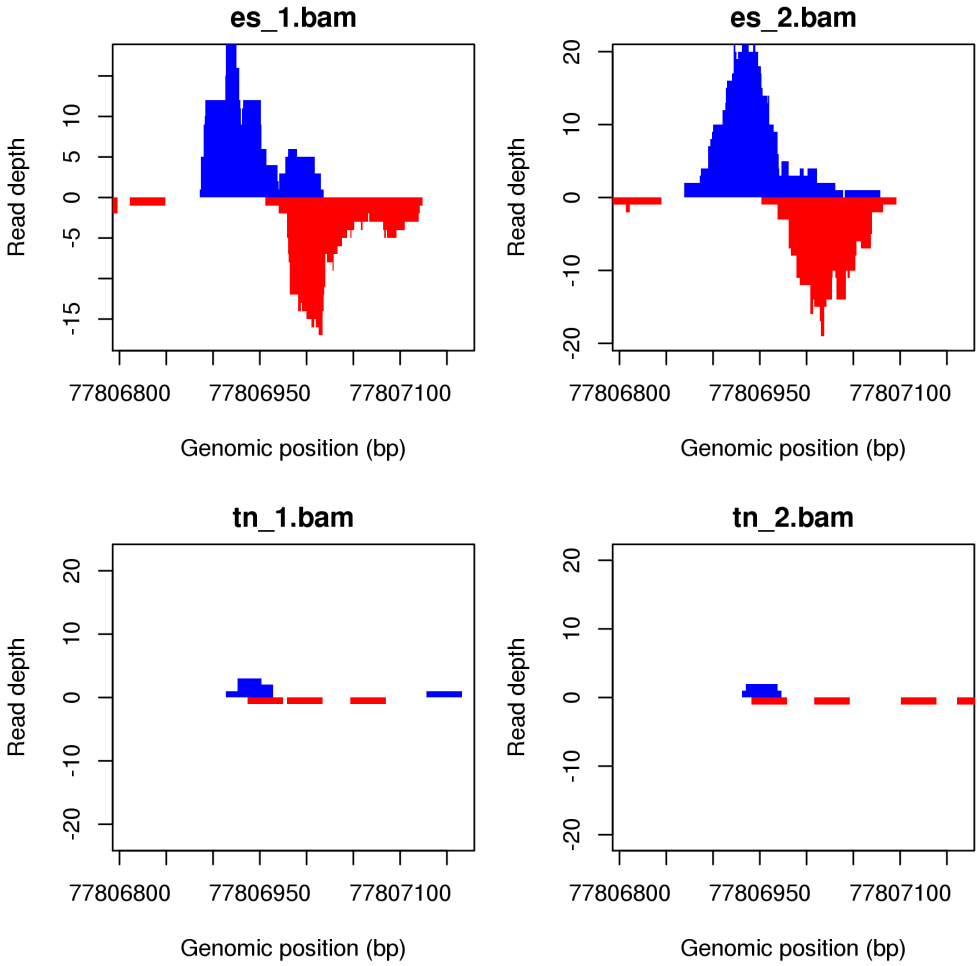
6.2 Looking at the results

It is then a simple matter to save the results. This is done here in the `*.tsv` format where all detail is preserved. Compression is just used to reduce the file size. Users can also save results to other formats using the rtracklayer package.

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

A quick and dirty inspection of the read depth around interesting features is also possible within R. 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 <- y$samples$lib.size*y$samples$norm.factors
> 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])
+ }
```


es_1.bam
Read depth
10
5
0
-5
-15
77806800
77806950
77807100
Genomic position (bp)
es_2.bam
Read depth
20
10
0
-10
-20
77806800
77806950
77807100
Genomic position (bp)
tn_1.bam
Read depth
20
10
0
-10
-20
77806800
77806950
77807100
Genomic position (bp)
tn_2.bam
Read depth
20
10
0
-10
-20
77806800
77806950
77807100
Genomic position (bp)

Chapter 7

Epilogue

What? You're still reading? The show's over, you can go home now. I suppose you're probably the type to stick around after movies to watch the credits. Well, I'll reward your enthusiasm with... a poem!

> 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.

7.1 Session information

```
> sessionInfo()

R version 3.0.0 (2013-04-03)
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=C                 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] org.Mm.eg.db_2.10.1  RSQLite_0.11.4       DBI_0.2-7
 [4] AnnotationDbi_1.24.0 Biobase_2.22.0       statmod_1.4.18
 [7] locfit_1.5-9.1       csaw_0.0.1           edgeR_4.5.16
[10] limma_3.18.3         Rsamtools_1.14.2     Biostrings_2.30.1
[13] GenomicRanges_1.14.3 XVector_0.2.0        IRanges_1.20.6
[16] BiocGenerics_0.8.0

loaded via a namespace (and not attached):
[1] bitops_1.0-6       grid_3.0.0         KernSmooth_2.23-10 lattice_0.20-24
[5] stats4_3.0.0       tools_3.0.0        zlibbioc_1.8.0
```

7.2 References

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 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.

M. S. Cheung, T. A. Down, I. Latorre, and J. Ahringer. Systematic bias in high-throughput sequencing data and its correction by BEADS. *Nucleic Acids Res.*, 39(15):e103, Aug 2011.

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.

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.

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. 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.

D. Risso, K. Schwartz, G. Sherlock, and S. Dudoit. GC-content normalization for RNA-Seq data. *BMC Bioinformatics*, 12:480, 2011.

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.

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.