diffHic: Differential analysis of Hi-C data User's Guide

Aaron T. L. Lun[1]

[1]The Walter and Eliza Hall Institute of Medical Research, Melbourne, Australia

October 13, 2017

Abstract

This document contains instructions on how to use *diffHic* for differential interaction analyses of Hi-C data. It covers counting of read pairs into bin pairs, filtering out low-abundance bin pairs, normalization of sample-specific trended biases, variance modelling and hypothesis testing, and summarization and visualization of bin pairs.

First edition: 12 December 2012
Last revised: 10 October 2017
Last compiled: 12 October 2017

Package

diffHic 1.9.8

Contents

Chapter 1

Introduction

1.1 Scope

This document describes the differential analysis of Hi-C data with the *diffHic* package. Differential interactions (DIs) are defined as interactions with significant changes in intensity between experimental conditions. DIs are identified in a statistically rigorous manner using methods in the *edgeR* package [1]. Knowledge of *edgeR* is useful but is not necessary for this guide.

1.2 How to get help

Most questions about individual functions should be answered by the documentation. For example, if you want to know more about `preparePairs`, you can bring up the documentation by typing `?preparePairs` or `help(preparePairs)` at the *R* prompt. Further detail on the methods or the theory can be found in the references at the bottom of each help page. The entire *diffHic* pipeline is also described in its own publication [2].

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 *diffHic* are best sent to the Bioconductor support site at https://support.bioconductor.org. Please send requests for general assistance and advice to the support site, rather than to the individual authors. Users posting to the support site for the first time may find it helpful to read the posting guide at http://www.bioconductor.org/help/support/posting-guide.

1.3 A brief description of Hi-C

The Hi-C protocol [3] is used to study chromatin organization by identifying pairwise interactions between two distinct genomic loci. Briefly, chromatin is cross-linked and digested with a restriction enzyme. This releases chromatin complexes into solution, where each complex contains multiple restriction fragments corresponding to interacting loci. Overhangs are filled in with biotin-labelled nucleotides to form blunt ends. Proximity ligation is then performed, which favors ligation between blunt ends in the same complex. The ligated DNA is sonicated and the sheared fragments containing ligation junctions are purified by a streptavidin pulldown. Paired-end sequencing is performed on the purified ligation products, and pairs of interacting loci are identified by mapping the paired reads. Of course, some caution is required due to the presence of non-specific ligation between blunt ends in different complexes.

1.4 Quick start

A typical differential analysis of Hi-C data is described below. For simplicity, assume that the BAM files have already been processed into "index" files in `input`. Let `design` contain the design matrix for this experiment. Also assume that the boundaries of the relevant restriction fragments are present in `fragments`. The code itself is split across several steps:

1. Converting BAM files to index files.
2. Counting read pairs into pairs of genomic bins.

```
library(diffHic)
param <- pairParam(fragments=fragments)
data <- squareCounts(input, width=1e6, param=param)
```

3. Filtering out uninteresting bin pairs.

```
library(edgeR)
keep <- aveLogCPM(asDGEList(data)) > 0
data <- data[keep,]
```

4. Normalizing for sample-specific biases.

```
data <- normOffsets(data, type="loess", se.out=TRUE)
y <- asDGEList(data)
```

5. Modelling biological variability between replicates.

```
y <- estimateDisp(y, design)
fit <- glmQLFit(y, design, robust=TRUE)
```

6. Testing for significant differences between groups.

```
result <- glmQLFTest(fit)
clusters <- diClusters(data, result$table, target=0.05,
                       cluster.args=list(tol=1))
```

In the various examples for this guide, data will be used from three studies. The first dataset examines the chromatin structure in K562 and GM06990 cell lines [3]. The second compares interaction intensities between wild-type and cohesin-deficient murine neural stem cells [4]. The final study compares ERG-overexpressing RWPE1 cells with a GFP-expressing control [5]. Obviously, readers will have to modify the code for their own analyses.

Chapter 2

Preparing index files from BAM files

This box will appear at the start of each chapter, describing the objects required from previous chapters. As we're starting out here, we don't really need anything.

2.1 A comment on aligning Hi-C libraries

In a typical Hi-C library, sequencing will occasionally be performed over the ligation junction between two restriction fragments. This forms a chimeric read that contains sequences from two distinct genomic loci. Here, correct alignment of the 5′ end of the chimeric read is more important. This is because the location of the 3′ end is already provided by mapping the 5′ end of the mate read. Direct application of alignment software will not be optimal as only one mapping location will be usually reported for each read. This means that the 5′ location will be ignored if the 3′ alignment is superior, e.g., longer or fewer mismatches.

Instead, chimeric alignment can be performed with approaches like iterative mapping [6] or read splitting [7]. In the latter, we define the ligation signature as the sequence that is obtained after ligation between blunt ends derived from cleaved restriction sites. For example, the *Hind*III enzyme cleaves at `AAGCTT` with a 4 bp overhang. This yields a signature sequence of `AAGCTAGCTT` upon ligation of blunt ends. The ligation signature in each chimeric read is identified with *cutadapt* [8], and the read is split into two segments at the center of the signature. Each segment is then aligned separately to the reference genome using *Bowtie2* [9]. Mapping by read splitting is performed in *diffHic* using a custom Python script:

```
system.file("python", "presplit_map.py", package="diffHic", mustWork=TRUE)
```

Users are strongly recommended to synchronize mate pair information and to mark duplicate read pairs in the resulting BAM file. This can be achieved using the various tools in the *Picard* software suite (http://broadinstitute.github.io/picard).

2.2 Matching mapped reads to restriction fragments

The Hi-C protocol is based on ligation between restriction fragments. Sequencing of the ligation product is performed to identify the interacting loci – or, more precisely, the two restriction fragments containing the interacting loci. The resolution of Hi-C data is inherently limited by the frequency of restriction sites and the size of the restriction fragments. Thus, it makes sense to report the read alignment location in terms of the restriction fragment to which that read was mapped. The boundaries of each restriction fragment can be obtained with the `cutGenome` function, as shown below for the human genome after digestion with the *Hin*dIII restriction enzyme (recognition site of `AAGCTT`, 5′ overhang of 4 bp).

```
library(BSgenome.Hsapiens.UCSC.hg19)
hs.frag <- cutGenome(BSgenome.Hsapiens.UCSC.hg19, "AAGCTT", 4)
hs.frag

## GRanges object with 846225 ranges and 0 metadata columns:
##                   seqnames         ranges strand
##                      <Rle>      <IRanges>  <Rle>
##        [1]            chr1 [    1, 16011]      *
##        [2]            chr1 [16008, 24575]      *
##        [3]            chr1 [24572, 27985]      *
##        [4]            chr1 [27982, 30433]      *
##        [5]            chr1 [30430, 32157]      *
##        ...             ...            ...    ...
##   [846221] chrUn_gl000249 [22854, 25904]      *
##   [846222] chrUn_gl000249 [25901, 31201]      *
##   [846223] chrUn_gl000249 [31198, 36757]      *
##   [846224] chrUn_gl000249 [36754, 36891]      *
##   [846225] chrUn_gl000249 [36888, 38502]      *
##   -------
##   seqinfo: 93 sequences from hg19 genome
```

These fragments should be stored in a *pairParam* object. The constructor below checks that the fragment ranges are properly ordered. Later, this object will also hold other parameters for counting. This simplifies coordination of the various steps in the *diffHic* pipeline, as the same *pairParam* object can be easily passed between different functions.

```
hs.param <- pairParam(hs.frag)
hs.param

## Genome contains 846225 restriction fragments across 93 chromosomes
## No discard regions are specified
## No limits on chromosomes for read extraction
## No cap on the read pairs per pair of restriction fragments
```

The `preparePairs` function matches the mapping location of each read to a restriction fragment in the reference genome. Mapping results for paired reads should be provided in a name-sorted BAM file. The function converts the read position into an index, pointing to the matching restriction fragment in `hs.frag`. The resulting pairs of indices are stored in an index file using the HDF5 format. The fragments to which the reads mapped are referred to as "anchors" – the larger index is defined as the first anchor fragment whereas the smaller is the second. This process is demonstrated using Hi-C data generated from GM06990 cells.

```
diagnostics <- preparePairs("SRR027957.bam", hs.param, file="SRR027957.h5",
                            dedup=TRUE, minq=10)
names(diagnostics)

## [1] "pairs"    "same.id"  "singles"  "chimeras"
```

The function itself returns a list of diagnostics showing the number of read pairs that are lost for various reasons. Of particular note is the removal of reads that are potential PCR duplicates with `dedup=TRUE`. This requires marking of the reads beforehand using an appropriate program such as *MarkDuplicates* from the *Picard* suite. Filtering on the minimum mapping quality score with `minq` is also recommended to remove spurious alignments.

```
diagnostics$pairs

##    total   marked filtered   mapped
##  7068675   103594  1532760  5460120
```

Read pairs mapping to the same restriction fragment provide little information on interactions between fragments [10]. Dangling ends are inward-facing read pairs that are mapped to the same fragment. These are uninformative as they are usually formed from sequencing of the restriction fragment prior to ligation. Self-circles are outward-facing read pairs that are formed when two ends of the same restriction fragment ligate to one another. Interactions within a fragment cannot be easily distinguished from these self-circularization events. Both structures are removed to avoid confusion in downstream steps.

```
diagnostics$same.id

##    dangling self.circle
##      425219      138248
```

2.3 Processing of chimeric reads

In the BAM file, chimeric reads are represented by two separate alignments for each read. Hard clipping is used to denote the length trimmed from each sequence in each alignment, and to determine which alignment corresponds to the 5′ or 3′ end of the read. Only the 5′ end(s) is used to determine the restriction fragment index for that read pair. The total number of chimeric read pairs will be reported by `preparePairs`, along with the number of pairs where both 5′ ends are mapped (`mapped`) and the nuber where both 5′ ends and at least one 3′ end are mapped (`multi`). Of course, the function will also work if the mapping location is only given for the 5′ end, though chimeric statistics will not be properly computed.

```
diagnostics$chimeras

##   total  mapped   multi invalid
## 2495159 1725843 1040864   67989
```

The proportion of invalid chimeric pairs is also calculated by `preparePairs`. Invalid pairs are those where the 3′ location of a chimeric read disagrees with the 5′ location of the mate. The invalid proportion can be used as an empirical measure of the mapping error rate - or, at least, the upper bound thereof, given that error rates are likely to be lower for longer, non-chimeric alignments. High error rates may be indicative of a fault in the mapping pipeline.

```
diagnostics$chimeras[["invalid"]]/diagnostics$chimeras[["multi"]]

## [1] 0.06531977
```

Invalid chimeric pairs can be discarded by setting `ichim=FALSE` in `preparePairs`. However, this is not recommended for routine analyses. Mapping errors for short 3′ ends may result in apparent invalidity and loss of the (otherwise correct) 5′ alignments.

2.4 Filtering artifactual read pairs

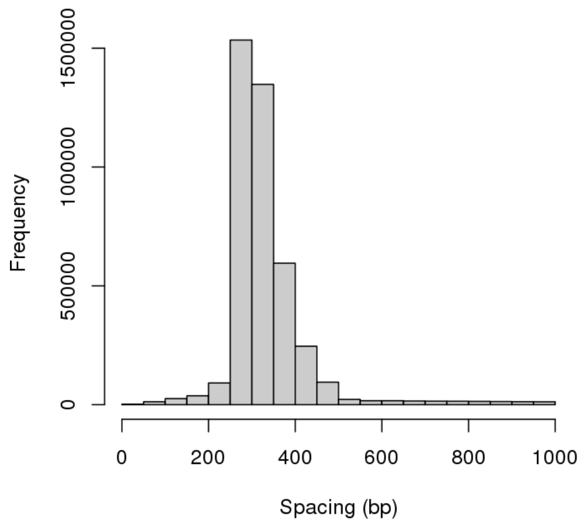
2.4.1 Reprocessing index files for quality control

The `prunePairs` function removes read pairs that correspond to artifacts in the Hi-C procedure. The returned vector contains the number of read pairs removed for each type of artifact. Values of `length`, `inward` and `outward` correspond to removal by `max.frag`, `min.inward` and `min.outward`, respectively. Retained read pairs are stored in another index file for later use.

```
min.inward <- 1000
min.outward <- 25000
prunePairs("SRR027957.h5", hs.param, file.out="SRR027957_trimmed.h5",
           max.frag=600, min.inward=min.inward, min.outward=min.outward)

##    total   length   inward  outward retained
##  4896653   870339    94644    82964  3860024
```

The `max.frag` argument removes read pairs where the inferred length of the sequencing fragment (i.e., the ligation product) is greater than a specified value. The length of the sequencing fragment is computed by summing, for both reads, the distance between the mapping location of the 5′ end and the nearest restriction site on the 3′ side. Excessively large lengths are indicative of off-site cleavage, i.e., where the restriction enzyme or some other agent cuts the DNA at a location other than the restriction site. While not completely uninformative, these are discarded as they are not expected from the Hi-C protocol. The threshold value can be chosen based on the size selection interval in library preparation, or by examining the distribution of inferred fragment lengths from `getPairData`.

```
diags <- getPairData("SRR027957.h5", hs.param)
hist(diags$length[diags$length < 1000], ylab="Frequency",
     xlab="Spacing (bp)", main="", col="grey80")
```



The insert size is defined as the linear distance between two paired reads on the same chromosome. The `min.inward` paramater removes inward-facing intra-chromosomal read pairs where the insert size is less than the specified value. The `min.outward` parameter does the same for outward-facing intra-chromosomal read pairs. This is designed to remove dangling ends or self-circles involving DNA fragments that have not been completely digested [11]. Such read pairs are technical artifacts that are (incorrectly) retained by `preparePairs`, as the two reads involved are mapped to different restriction fragments. Appropriate thresholds for both parameters can be determined using strand orientation plots.

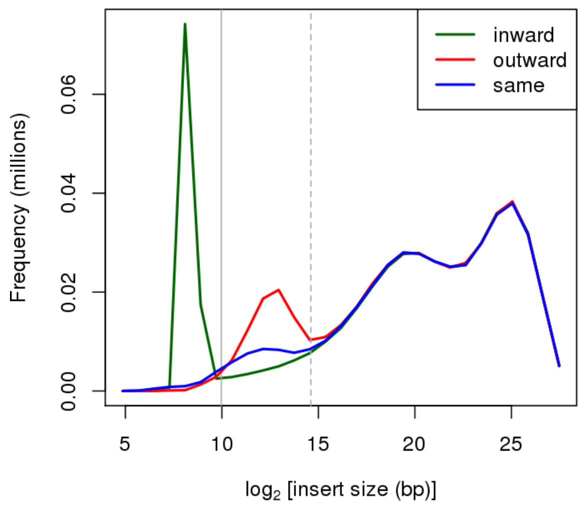
2.4.2 Setting size thresholds with strand orientation plots

The strand orientation for a read pair refers to the combination of strands for the two alignments. These are stored as flags where setting `0x1` or `0x2` means that the read on the first or second anchor fragment, respectively, is mapped on the reverse strand. If different pieces of DNA were randomly ligated together, one would expect to observe equal proportions of all strand orientations. This can be tested by examining the distribution of strand orientations for inter-chromosomal read pairs. Each orientation is equally represented across these read pairs, which is expected as different chromosomes are distinct pieces of DNA.

```
intra <- !is.na(diags$insert)
table(diags$orientation[!intra])

##
##      0      1      2      3
## 768247 764801 764579 764536
```

This can be repeated for intra-chromosomal read pairs, by plotting the distribution of insert sizes for each strand orientation [11]. The two same-strand distributions are averaged for convenience. At high insert sizes, the distributions will converge for all strand orientations. This is consistent with random ligation between two separate restriction fragments. At lower insert sizes, spikes are observed in the ouward- and inward-facing distributions due to self-circularization and dangling ends, respectively. Thresholds should be chosen in `prunePairs` to remove these spikes, as represented by the grey lines.

```
# Constructing the histograms for each orientation.
llinsert <- log2(diags$insert + 1L)
intra <- !is.na(llinsert)
breaks <- seq(min(llinsert[intra]), max(llinsert[intra]), length.out=30)
inward <- hist(llinsert[diags$orientation==1L], plot=FALSE, breaks=breaks)
outward <- hist(llinsert[diags$orientation==2L] ,plot=FALSE, breaks=breaks)
samestr <- hist(llinsert[diags$orientation==0L | diags$orientation==3L],
                plot=FALSE, breaks=breaks)
samestr$counts <- samestr$counts/2

# Setting up the axis limits.
ymax <- max(inward$counts, outward$counts, samestr$counts)/1e6
xmax <- max(inward$mids, outward$mids, samestr$mids)
xmin <- min(inward$mids, outward$mids, samestr$mids)

# Making a plot with all orientations overlaid.
plot(0,0,type="n", xlim=c(xmin, xmax), ylim=c(0, ymax),
     xlab=expression(log[2]~"[insert size (bp)]"), ylab="Frequency (millions)")
lines(inward$mids, inward$counts/1e6, col="darkgreen", lwd=2)
abline(v=log2(min.inward), col="darkgrey")
lines(outward$mids, outward$counts/1e6, col="red", lwd=2)
abline(v=log2(min.outward), col="darkgrey", lty=2)
lines(samestr$mids, samestr$counts/1e6, col="blue", lwd=2)
legend("topright", c("inward", "outward", "same"),
       col=c("darkgreen", "red", "blue"), lwd=2)
```



As an aside, the position of the spikes in the above plots can be used to estimate some fragment lengths. The x-coordinate of the outward-facing spike represents the average length of the DNA fragments after restriction digestion. This is useful as it provides a lower bound on the spatial resolution of any given Hi-C experiment. The position of the inward-facing spike represents the average length of the fragments after sonication. This should be lower than the size selection thresholds used in library preparation.

2.5 Merging technical replicates

Hi-C experiments often involve deep sequencing as read pairs are sparsely distributed across the set of possible interactions. As a result, multiple index files may be generated from multiple technical replicates of a single Hi-C library. These can be merged together using the `mergePairs` function prior to downstream processing. This is equivalent to summing the counts for each pair of restriction fragment indices, and is valid if one assumes Poisson sampling for each sequencing run [12]. An example is provided below that merges several technical replicates for a Hi-C library generated from GM06990 cells [3].

```
prepped <- preparePairs("SRR027958.bam", hs.param, file="SRR027958.h5",
                        dedup=TRUE, minq=10)
counted <- prunePairs("SRR027958.h5", hs.param, file.out="SRR027958_trimmed.h5",
                      max.frag=600, min.inward=min.inward,
                      min.outward=min.outward)
mergePairs(files=c("SRR027957_trimmed.h5", "SRR027958_trimmed.h5"), "merged.h5")
```

In addition, any Hi-C dataset that is processed manually by the user can be stored in an index file using the `savePairs` function. This takes a dataframe with the first and second anchor indices, as well as any additional information that might be useful. The idea is to provide an entry point into the *diffHic* analysis from other pipelines. If the dataset is too large, one can save chunks at a time before merging them all together with `mergePairs`.

```
anchor1.id <- as.integer(runif(100, 1, length(hs.param$fragments)))
anchor2.id <- as.integer(runif(100, 1, length(hs.param$fragments)))
dummy <- data.frame(anchor1.id, anchor2.id, other.data=runif(100))
savePairs(dummy, "example.h5", hs.param)
```

For full compatibility, users should include the alignment positions and lengths for each read pair as `anchorX.pos` and `anchorX.len`, where `X` is 1 or 2 for each of the paired reads. The alignment position refers to the 1-based coordinate of the left-most base of the alignment. The alignment length refers to the span of the alignment relative to the reference, and should be negative for alignments on the reverse strand. This information will be used in downstream *diffHic* functions, such as read counting around blacklisted regions.

2.6 Handling DNase Hi-C experiments

DNase Hi-C is a variant of the standard protocol whereby the DNase I enzyme is used to fragment the genome instead of restriction enzymes [13]. Random fragmentation provides resolution beyond that of individual restriction fragments. However, cleavage sites for DNase I cannot be easily predicted to construct `param$fragments`. Fragment indices have no meaning here because there are no restriction fragments for reads to be assigned to.

Instead, *diffHic* handles this type of data by operating directly on the alignment position of each read. This reflects the theoretical base-pair resolution of the data during quantification. To indicate that the reads come from a DNase Hi-C experiment, an empty *GRanges* object should be supplied as the `fragments` in the *pairParam* object. Most *diffHic* functions will detect this and behave appropriately to exploit the improved resolution. An example of this approach is shown below, where the SRR027957 library is treated as a DNase Hi-C sample.

```
seg.frags <- emptyGenome(BSgenome.Hsapiens.UCSC.hg19)
preparePairs("SRR027957.bam", pairParam(seg.frags), file="SRR027957.h5",
             dedup=TRUE, minq=10)

## $pairs
##    total   marked filtered   mapped
##  7068675   103594  1532760  5460120
##
## $singles
## [1] 0
##
## $chimeras
##   total  mapped   multi invalid
## 2495159 1779050 1073948   41582
```

Unlike `preparePairs`, no diagnostic information regarding self-circles or dangling ends is reported here. Such metrics are irrelevant when restriction fragments are not involved. Similarly, the `length` field in the output of `getPairData` is set to `NA` and should be ignored. This is because the length of the sequencing fragment cannot be generally computed without knowledge of the fragmentation sites. As a result, the `max.frag` argument should also be set to `NA` in `prunePairs`. Metrics for inward- and outward-facing read pairs are unaffected, as these are computed independently of the fragments. See the documentation for individual functions to determine if/how their behaviour changes for DNase Hi-C data.

Users should also note that the alignment script described in Section 2.1 is not appropriate for DNase Hi-C experiments. This approach is based on splitting chimeric reads at the ligation signature, which is constructed from the recognition site of a restriction enzyme. The sequence around the ligation junction is not well-defined when DNase I is used for cleavage. Instead, alignment programs should be used that can directly handle chimeric reads with arbitrary breakpoints in the genome, e.g., *BWA* [14]. A Python script for iterative mapping [6] is also provided for this purpose.

```
system.file("python", "iter_map.py", package="diffHic", mustWork=TRUE)
```

Chapter 3

Counting read pairs into interactions

A different dataset will be used here, so we don't need anything from the last chapter. Horses for courses; this dataset's a lot nicer for detecting differential interactions.

3.1 Overview

Prior to any statistical analysis, the read pairs in a Hi-C library must be summarized into a count for each interaction. This count is used as an experimental measure of the interaction intensity. Specifically, each pairwise interaction is parameterized by two genomic intervals representing the interacting loci. The count for that interaction is defined as the number of read pairs with one read mapping to each of the intervals. Counting is performed for each sample in the dataset, such that each interaction is associated with a set of counts.

The interaction space is defined as the genome-by-genome space over which the read pairs are distributed. Recall that each paired read is assigned to a restriction fragment index. The interaction space contains all index pairs (x, y) for $x, y \in [1..N]$, where $x \geq y$ and N is the number of restriction fragments in the genome. This can be visualized as the triangular space between $y = x$, $y = 0$ and $x = N$ on the Cartesian plane. A rectangular area in the interaction space represents a pairwise interaction between the genomic intervals spanned by the two adjacent sides of that rectangle. The number of read pairs in this area is used as the count for the corresponding interaction. Non-rectangular areas can also represent interactions, but these are more difficult to interpret and will not be considered here.

The examples shown here will use the neural stem cell dataset [4] in which wild-type cells are compared to cohesin-deficient cells. Read processing has already been performed to construct an index file for each library. Some additional work is required to obtain the restriction fragment coordinates for the *Hin*dIII-digested mouse genome.

```
library(BSgenome.Mmusculus.UCSC.mm10)
mm.frag <- cutGenome(BSgenome.Mmusculus.UCSC.mm10, "AAGCTT", 4)
input <- c("merged_flox_1.h5", "merged_flox_2.h5",
           "merged_ko_1.h5", "merged_ko_2.h5")
```

3.2 Counting into bin pairs

3.2.1 Overview

Here, the genome is partitioned into contiguous non-overlapping bins of constant size. Each interaction is defined as a pair of these bins. This approach avoids the need for prior knowledge of the loci of interest when summarizing Hi-C counts. Counting of read pairs between paired bins is performed for multiple libraries using the `squareCounts` function.

```
bin.size <- 1e6
mm.param <- pairParam(mm.frag)
data <- squareCounts(input, mm.param, width=bin.size, filter=1)
data

## class: InteractionSet
## dim: 3319100 4
## metadata(2): param width
## assays(1): counts
## rownames: NULL
## rowData names(0):
## colnames: NULL
## colData names(1): totals
## type: ReverseStrictGInteractions
## regions: 2739
```

This generates an *InteractionSet* object containing information for multiple genomic interactions [15]. Each row of the object corresponds to an interaction, i.e., bin pair, while each column represents a library. For each interaction, the coordinates of its two "anchor" bins are returned. The first/second anchor notation is used for these bins, whereby the first anchor bin is that with the "higher" genomic coordinate.

```
head(anchors(data, type="first"))

## GRanges object with 6 ranges and 1 metadata column:
##       seqnames             ranges strand |    nfrags
##          <Rle>          <IRanges>  <Rle> | <integer>
##   [1]     chr1 [3004106, 4000741]      * |       389
##   [2]     chr1 [3004106, 4000741]      * |       389
##   [3]     chr1 [4000738, 5001375]      * |       334
##   [4]     chr1 [4000738, 5001375]      * |       334
##   [5]     chr1 [4000738, 5001375]      * |       334
##   [6]     chr1 [5001372, 5997485]      * |       340
##   -------
##   seqinfo: 66 sequences from an unspecified genome

head(anchors(data, type="second"))

## GRanges object with 6 ranges and 1 metadata column:
##       seqnames             ranges strand |    nfrags
##          <Rle>          <IRanges>  <Rle> | <integer>
##   [1]     chr1 [      1, 3004109]      * |         1
##   [2]     chr1 [3004106, 4000741]      * |       389
##   [3]     chr1 [      1, 3004109]      * |         1
##   [4]     chr1 [3004106, 4000741]      * |       389
##   [5]     chr1 [4000738, 5001375]      * |       334
##   [6]     chr1 [      1, 3004109]      * |         1
##   -------
##   seqinfo: 66 sequences from an unspecified genome
```

The returned *InteractionSet* object contains a count matrix with the number of read pairs for each interaction in each library. It also contains a `totals` vector in the `colData`, which stores the total number of read pairs for each library.

```
head(assay(data))

##       [,1]  [,2]  [,3]  [,4]
## [1,]    83    48    33    19
## [2,] 21332 17151 12894 12357
## [3,]    20    20    17     6
## [4,]  8215  7023  4399  4237
## [5,] 14729 12460  8985  8443
## [6,]     7     2     3     0

data$totals

## [1] 85786306 74685186 60860491 54596160
```

Bin pairs can also be filtered to remove those with to a count sum below `filter`. This removes uninformative bin pairs with very few read pairs, and reduces the memory footprint of the function. A higher value of `filter` may be necessary for analyses of large datasets in limited memory. More sophisticated filtering strategies are discussed in Chapter 4.

3.2.2 Choosing a bin width

The `width` of the bin is specified in base pairs and determines the spatial resolution of the analysis. Smaller bins will have greater spatial resolution as adjacent features can be distinguished in the interaction space. Larger bins will have greater counts as a larger area is used to collect read pairs. Optimal summarization will not be achieved if bins are too small or too large to capture the (changes in) intensity of the underlying interactions.

For this analysis, 1 Mbp bins are used to capture broad structural features. This is also useful for demonstration purposes, as the counts are large enough for clear manifestation of biases in later chapters. Of course, *diffHic* is more than capable of handling smaller sizes (e.g., below 20 kbp) for higher-resolution analyses of looping interactions between specific elements. In such cases, `filter` should be increased to avoid excessive memory usage.

```
head(regions(data))

## GRanges object with 6 ranges and 1 metadata column:
##       seqnames             ranges strand |    nfrags
##          <Rle>          <IRanges>  <Rle> | <integer>
##   [1]     chr1 [      1, 3004109]      * |         1
##   [2]     chr1 [3004106, 4000741]      * |       389
##   [3]     chr1 [4000738, 5001375]      * |       334
##   [4]     chr1 [5001372, 5997485]      * |       340
##   [5]     chr1 [5997482, 7000260]      * |       342
##   [6]     chr1 [7000257, 8000015]      * |       349
##   -------
##   seqinfo: 66 sequences from an unspecified genome
```

The boundary of each bin is rounded to the closest restriction site in `squareCounts`. This is due to the inherent limits on spatial resolution in a Hi-C experiment. The number of restriction fragments in each bin is recorded in the `nfrags` field of the metadata.

Determination of the ideal bin size is not trivial as the features of interest are not usually known in advance. Instead, repeated analyses with multiple bin sizes are recommended. This provides some robustness to the choice of bin size. Sharp interactions can be detected by pairs of smaller bins while diffuse interactions can be detected by larger bin pairs. See Section 7.3 for more information on consolidating results from multiple bin sizes.

3.3 Counting with pre-defined regions

3.3.1 Connecting counts between pairs of regions

For some studies, prior knowledge about the regions of interest may be available. For example, a researcher may be interested in examining interactions between genes. The coordinates can be obtained from existing annotation, as shown below for the mouse genome. Other pre-specified regions can also be used, e.g., known enhancers or protein binding sites.

```
library(TxDb.Mmusculus.UCSC.mm10.knownGene)
gene.body <- genes(TxDb.Mmusculus.UCSC.mm10.knownGene)
strand(gene.body) <- "*" # Removing strand info, for simplicity.
```

Counting is directly performed for these defined regions using the `connectCounts` function. Interactions are defined between each pair of regions in the pre-specified set. This may be easier to interpret than pairs of bins if the interacting regions have some biological significance. The count matrix and the vector of totals are defined as previously described.

```
redata <- connectCounts(input, mm.param, regions=gene.body)
redata

## class: InteractionSet
## dim: 12926724 4
## metadata(1): param
## assays(1): counts
## rownames: NULL
## rowData names(0):
## colnames: NULL
## colData names(1): totals
## type: ReverseStrictGInteractions
## regions: 24044
```

Again, first/second anchor notation applies whereby the interval with the larger start coordinate in the genome is defined as the first anchor. Note that the anchor may not have a larger end coordinate if the supplied `regions` are nested. In addition, each region is rounded to the nearest restriction site. Resorting is also performed, though the indices of the original regions can be found in the metadata as `original` if back-referencing is necessary.

```
head(regions(redata))

## GRanges object with 6 ranges and 2 metadata columns:
##          seqnames             ranges strand |    nfrags  original
##             <Rle>          <IRanges>  <Rle> | <integer> <integer>
##   497097     chr1 [3211067, 3675240]      * |       197     15372
##    19888     chr1 [4286646, 4409818]      * |        49      7094
##    20671     chr1 [4487488, 4498343]      * |         2      7482
##    27395     chr1 [4772601, 4786029]      * |         3     13110
##    18777     chr1 [4802092, 4847111]      * |         8      6523
##    21399     chr1 [4856126, 4899085]      * |        16      8171
##   -------
##   seqinfo: 66 sequences (1 circular) from mm10 genome
```

One obvious limitation of this approach is that interactions involving unspecified regions will be ignored. This is obviously problematic when searching for novel interacting loci. Another issue is that the width of the regions cannot be easily changed. This means that the compromise between spatial resolution and count size cannot be tuned. For example, interactions will not be detected around smaller genes as the counts will be too small. Conversely, interactions between distinct loci within a large gene body will not be resolved.

3.3.2 Connecting counts between two sets of regions

The `connectCounts` function can also be used to identify interactions between two sets of regions, by specifying a value for the `second.regions` argument. This only considers interactions between one entry in `regions` and another in `second.regions`. This differs from the standard application of the function, which would consider an interaction between any pair of entries in `regions`. If an integer scalar is supplied as `second.regions`, the second set is automatically defined as contiguous bins of that size across the genome.

The use of `second.regions` is particularly useful in cases where there are defined "viewpoints" of interest, e.g., 4C-seq, Capture-C. These viewpoints can be specified in `regions`, as shown below for a set of mock probe locations for a hypothetical Capture-C experiment [16]. Specifically, the viewpoint is defined as a 100 kbp bin centred at each capture probe. The interaction profile across the rest of the genome can then be extracted by setting `second.regions` to some bin size. In this case, 100 kbp bins are used.

```
probe.loc <- GRanges(c("chr1", "chr2", "chr3"),
                     IRanges(c(1e7, 2e7, 3e7), c(1e7, 2e7, 3e7)))
viewpoints <- resize(probe.loc, fix="center", width=1e5)
viewdata <- connectCounts(input, mm.param, regions=viewpoints,
                          second.regions=1e5)
head(anchors(viewdata, type="first"))

## GRanges object with 6 ranges and 3 metadata columns:
##       seqnames              ranges strand |    nfrags is.second  original
##          <Rle>           <IRanges>  <Rle> | <integer> <logical> <integer>
##   [1]     chr1 [9945397, 10054278]      * |        38         0         1
##   [2]     chr1 [9945397, 10054278]      * |        38         0         1
##   [3]     chr1 [9945397, 10054278]      * |        38         0         1
##   [4]     chr1 [9945397, 10054278]      * |        38         0         1
##   [5]     chr1 [9945397, 10054278]      * |        38         0         1
##   [6]     chr1 [9945397, 10054278]      * |        38         0         1
##   -------
##   seqinfo: 66 sequences from an unspecified genome

head(anchors(viewdata, type="second"))

## GRanges object with 6 ranges and 3 metadata columns:
##       seqnames             ranges strand |    nfrags is.second  original
##          <Rle>          <IRanges>  <Rle> | <integer> <logical> <integer>
##   [1]     chr1 [      1, 3004109]      * |         1         1      <NA>
##   [2]     chr1 [3004106, 3100835]      * |        34         1      <NA>
##   [3]     chr1 [3100832, 3194461]      * |        30         1      <NA>
##   [4]     chr1 [3194458, 3301641]      * |        41         1      <NA>
##   [5]     chr1 [3301638, 3399158]      * |        52         1      <NA>
##   [6]     chr1 [3399155, 3501374]      * |        39         1      <NA>
##   -------
##   seqinfo: 66 sequences from an unspecified genome
```

As these results demonstrate, interactions are only considered if exactly one interacting locus is from the specified `regions`. The identity of the other locus (i.e., from `second.regions`) can be determined based on the `is.second` field in the *GRanges* object. This approach avoids loading irrelevant interactions when only specific viewpoints are of interest.

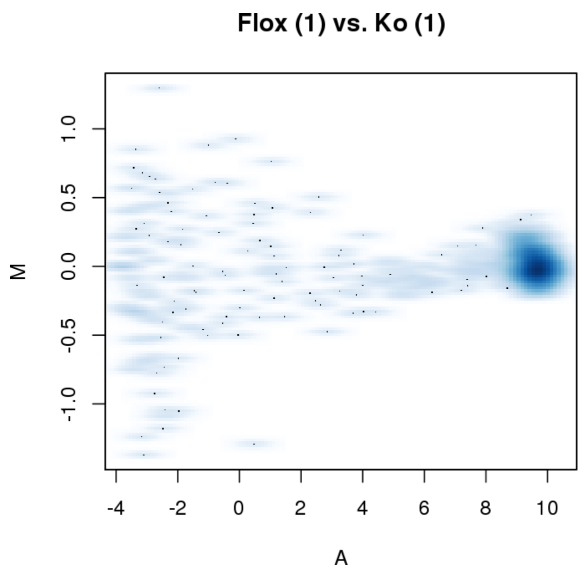
3.4 Counting into single bins

The `marginalCounts` function counts the number of reads mapped inside each bin. This effectively treats each Hi-C library as single-end data to quantify the genomic coverage of each bin. One can use these "marginal" counts to determine whether there are systematic differences in coverage between libraries for a given bin. This implies that copy number variations are present, which may confound detection of differential interactions.

```
margin.data <- marginCounts(input, mm.param, width=bin.size)
margin.data

## class: RangedSummarizedExperiment
## dim: 2739 4
## metadata(1): param
## assays(1): counts
## rownames: NULL
## rowData names(1): nfrags
## colnames: NULL
## colData names(1): totals
```

Each row of the *RangedSummarizedExperiment* contains the marginal read counts for a genomic bin. For this dataset, there are no major changes in coverage for the vast majority of bins. The most extreme events occur at low abundances and are unlikely to be precise. This suggests that a direct comparison of interaction intensities will be valid. Remedial action in the presence of copy number changes is not trivial and will be discussed in Section 5.4.

```
adjc <- cpm(asDGEList(margin.data), log=TRUE, prior.count=5)
smoothScatter(0.5*(adjc[,1]+adjc[,3]), adjc[,1]-adjc[,3],
              xlab="A", ylab="M", main="Flox (1) vs. Ko (1)")
```



3.5 Additional parameter options

3.5.1 Restricting the input chromosomes

Users can restrict counting to particular chromosomes by setting the `restrict` slot in the *pairParam* object. This is useful to ensure that only interactions between relevant chromosomes are loaded. Sequences such as the mitochondrial genome, unassigned contigs or random chromosome segments can be ignored in routine analyses.

```
new.param <- reform(mm.param, restrict=c("chr1", "chr2"))
new.param

## Genome contains 851636 restriction fragments across 66 chromosomes
## No discard regions are specified
## Read extraction is limited to 2 chromosomes
## No cap on the read pairs per pair of restriction fragments
```

In addition, if `restrict` is a n-by-2 matrix, count loading will be limited to the read pairs that are mapped between the n specified pairs of chromosomes. The example below considers all read pairs mapped between chromosomes 2 and 19. This feature is useful when memory is limited, as each pair of chromosomes can be loaded and analyzed separately.

```
new.param <- reform(mm.param, restrict=cbind("chr2", "chr19"))
new.param

## Genome contains 851636 restriction fragments across 66 chromosomes
## No discard regions are specified
## Read extraction is limited to pairs between:
##  'chr2' and 'chr19'
## No cap on the read pairs per pair of restriction fragments
```

3.5.2 Specifying regions to ignore

Users can also discard alignments that lie within blacklisted regions by setting the `discard` slot. The aim is to eliminate reads within known repeat regions. Such regions are problematic, as reads from several repeat units in the real genome may be collated into a single representative unit in the genome build. This results in a sharp, spurious spike in interaction intensity. The problem is exacerbated by different repeat copy numbers between biological conditions, resulting in spurious differential interactions due to changes in coverage. Removal of reads in these repeats may be necessary to obtain reliable results.

```
dummy.repeat <- GRanges("chr1", IRanges(10000, 1000000))
new.param <- reform(mm.param, discard=dummy.repeat)
new.param

## Genome contains 851636 restriction fragments across 66 chromosomes
## 1 region specified in which alignments are discarded
## No limits on chromosomes for read extraction
## No cap on the read pairs per pair of restriction fragments
```

Coordinates of annotated repeats can be obtained from several different sources. A curated blacklist of problematic regions is available from the ENCODE project [17], and can be obtained at https://sites.google.com/site/anshulkundaje/projects/blacklists. This list is constructed empirically from the ENCODE datasets and includes obvious offenders like telomeres, microsatellites and some rDNA genes. Alternatively, repeats can be predicted from the genome sequence using software like RepeatMasker. These calls are available from the UCSC website (e.g., hgdownload.soe.ucsc.edu/goldenPath/mm10/bigZips/chromOut.tar.gz for mouse) or they can be extracted from an appropriate masked *BSgenome* object. Experience suggests that the ENCODE blacklist is generally preferable. Repeat predictions tend to be aggressive such that too much of the genome (and interactions therein) will be discarded.

3.5.3 Capping the read pairs per restriction fragment pair

Incomplete removal of PCR duplicates or read pairs in repeat regions may result in spikes of read pairs within the interaction space. The effect of these artifacts can be mitigated by capping the number of read pairs associated with each pair of restriction fragments. This is done by specifying a value for the `cap` slot. Diffuse interactions should not be affected, as the associated read pairs will be distributed sparsely across many fragment pairs. More caution is required if sharp interactions are present, i.e., interactions between 5 - 10 kbp regions.

```
new.param <- reform(mm.param, cap=5)
new.param

## Genome contains 851636 restriction fragments across 66 chromosomes
## No discard regions are specified
## No limits on chromosomes for read extraction
## Cap of 5 on the read pairs per pair of restriction fragments
```

3.6 Loading counts from existing count matrices

It is possible to use counts from existing count matrices, by using coercion methods available to *ContactMatrix* objects from the *InteractionSet* package [15]. Assume that there are two *ContactMatrix* objects (`cm1` and `cm2`) spanning the same region of the binned interaction space. The `mergeCMs` function converts these two objects into a single *InteractionSet* object for use in the *diffHic* analysis pipeline. Only bin pairs with a count sum greater than 10 are retained, analogous to the output obtained with a `squareCounts` call with `filter=10`.

```
data.com <- mergeCMs(cm1, cm2, filter=10)
```

This procedure facilitates input from other Hi-C data processing pipelines that return count matrices rather than BAM files. It is easily extended to datasets involving more than two samples by simply including additional *ContactMatrix* objects in the `mergeCMs` call.

3.7 Summary

Counting into bin pairs is the most general method for quantifying interaction intensity. It does not require any prior knowledge regarding the regions of interest. The bin size can be easily adjusted to obtain the desired spatial resolution. It is also easier/safer to compare between bin pairs (e.g., during filtering) when each bin is roughly of the same size. Thus, bin-based counting will be the method of choice for the rest of this guide.

For simplicity, all counting steps will be performed here with the default settings, i.e., no values for `restrict`, `discard` or `cap`. However, users are encouraged to use non-default values if it can improve the outcome of their analyses. Non-default values should be recorded in a single *pairParam* object for consistent use across all functions in the *diffHic* pipeline. This ensures that the same read pairs will always be extracted from each index file.

Chapter 4

Filtering out uninteresting interactions

Here, we'll need the `data` object that was loaded in the previous chapter. We'll also need `bin.size` and `mm.param`, as well as the file names in `input`.

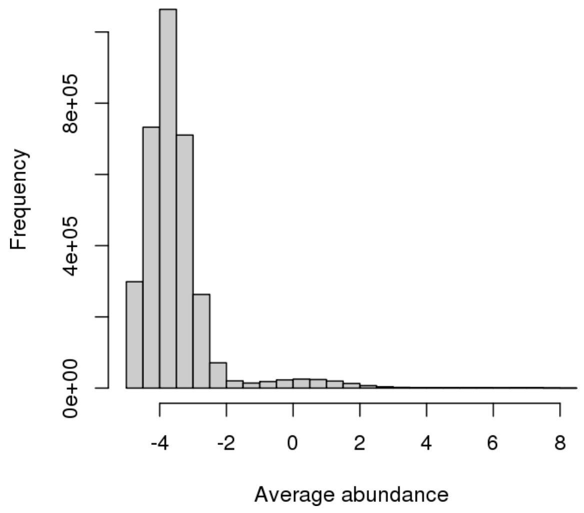
4.1 Overview

4.1.1 Computing the average abundance in a NB model

Filtering is often performed to remove uninteresting features in analyses of high-throughput experiments. This reduces the severity of the multiplicity correction and increases power among the remaining tests. The filter statistic should be independent of the p-value under the null hypothesis, but correlated to the p-value under the alternative [18]. The aim is to enrich for false nulls without affecting type I error for the true nulls.

Assume that the counts for each bin pair are sampled from the negative binomial (NB) distribution. In this model, the overall NB mean across all libraries is (probably) an independent filter statistic. The log-mean-per-million is known as the average abundance and can be computed with the `aveLogCPM` function in *edgeR* [19].

```
library(edgeR)
ave.ab <- aveLogCPM(asDGEList(data))
hist(ave.ab, xlab="Average abundance", col="grey80", main="")
```



Any bin pair with an average abundance less than a specified threshold value will be discarded. At the very least, the threshold should be chosen to filter out bin pairs with very low absolute counts. This is because these bin pairs will never have sufficient evidence to reject the null hypothesis. The discreteness of low counts will also reduce the accuracy of approximations that are used in normalization and statistical modelling. The example below identifies those bin pairs where the overall NB mean across all samples is not less than 5.

```
count.keep <- ave.ab >= aveLogCPM(5, lib.size=mean(data$totals))
summary(count.keep)

##    Mode   FALSE    TRUE
## logical 2433389  885711
```

The `count.keep` vector is then used to subset the *InteractionSet* object. The resulting `dummy` object will contain only the interesting bin pairs for downstream analysis. The same procedure can be used for any logical or integer vector that specifies the bin pairs to be retained.

```
dummy <- data[count.keep,]
```

This count-based approach is fairly objective yet is still effective, i.e., it removes a large number of bin pairs that are likely to be uninteresting. However, it will be less useful with greater sequencing depth where all bin pairs will have higher counts. More sophisticated strategies can be implemented where the choice of threshold is motivated by some understanding of the Hi-C protocol. These strategies will be described in the rest of this chapter.

4.2 Directly removing low-abundance interactions

4.2.1 Computing the threshold from inter-chromosomal counts

The simplest definition of an "uninteresting" interaction is that resulting from non-specific ligation. These are represented by low-abundance bin pairs where no underlying contact is present to drive ligation between the corresponding bins. Any changes in the counts for these bin pairs are uninteresting and are ignored. In particular, the filter threshold can be defined by mandating some minimum fold change above the level of non-specific ligation.

The magnitude of non-specific ligation is empirically estimated by assuming that most inter-chromosomal contacts are not genuine. This is reasonable given that most chromosomes are arranged in self-interacting territories [20]. The median abundance across inter-chromosomal bin pairs is used as an estimate of the non-specific ligation rate. Here, filtering is performed to retain only those bin pairs with abundances that are at least 5-fold higher than this estimate. This aims to remove the majority of uninteresting bin pairs.

```
direct <- filterDirect(data)
direct$threshold

## [1] -3.845944

direct.keep <- direct$abundances > log2(5) + direct$threshold
summary(direct.keep)

##    Mode   FALSE    TRUE
## logical 3159187  159913
```

The `direct.keep` vector can then be used to filter `data`, as shown previously. This approach is named here as "direct" filtering, as the average abundance for each bin pair is directly compared against a fixed threshold value. In practice, the direct filter can be combined with `count.keep` to ensure that the retained bin pairs have large absolute counts.

```
direct.keep2 <- direct.keep & count.keep
```

4.2.2 Computing the threshold from larger bin pairs

The procedure above assumes that no additional filtering has been performed during count loading with `squareCounts`, i.e., `filter` is set to unity. Any pre-filtering that removes low-abundance bin pairs will inflate the median and lead to overestimation of the filter threshold. However, it may not be practical to load counts without pre-filtering. For example, at small bin sizes, too many non-empty bin pairs may be present to fit into memory. Counts that are too small will also yield imprecise estimates of the background ligation rate.

To overcome this, counts are loaded for a larger bin size without pre-filtering. This reduces memory usage as the interaction space is partitioned into fewer bin pairs. The filter threshold is estimated from the inter-chromosomal interactions as previously described, exploiting the presence of large counts for large bin sizes to achieve precise estimates. The estimated threshold for the larger bin pairs is then converted into a corresponding threshold for the original (smaller) bin pairs, after adjusting for the differences in bin sizes.

To illustrate, imagine that a bin size of 100 kbp is of interest. We load the counts for the smaller bin pairs, using a reasonably large `filter` to avoid excessive memory consumption.

```
new.bin.size <- 1e5
smaller.data <- squareCounts(input, mm.param, width=new.bin.size, filter=20)
```

Direct filtering of these smaller bin pairs is performed using `filterDirect`, using the larger (1 Mbp) bin pairs to estimate the filter threshold. This is done by passing the unfiltered counts for the larger bin pairs in `data` as the `reference` parameter. The returned statistics can then be used to directly filter the smaller bin pairs. Here, a minimum 5-fold change over the threshold is required for the retention of each 100 Mbp bin pair.

```
direct <- filterDirect(smaller.data, reference=data)
direct$threshold

## [1] -5.087542

small.keep <- direct$abundances > direct$threshold + log2(5)
summary(small.keep)

##    Mode   FALSE    TRUE
## logical  426056  634789
```

4.3 Filtering as a function of interaction distance

4.3.1 Computing the threshold directly from the bin pair counts

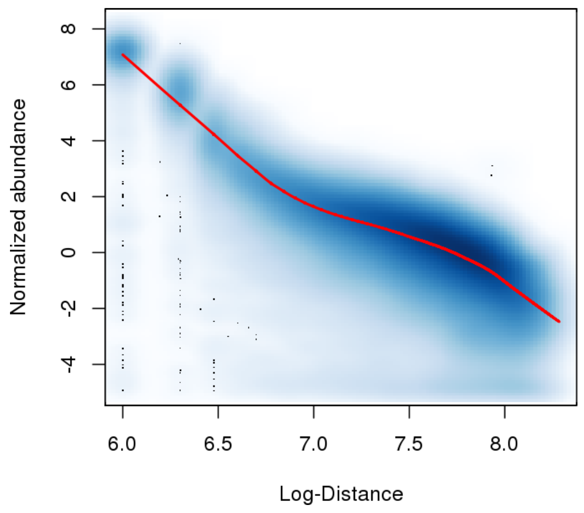
A more complex filter involves adjusting the threshold according to the distance between the bins in each bin pair. In typical Hi-C datasets, larger counts are observed at lower interaction distances. This is probably driven by non-specific compaction of chromatin into a 3D "globule" conformation. If such compaction is uninteresting, a concomitantly higher threshold is necessary to offset the increase in counts for these local interactions [21].

In the trended strategy, a trend is fitted to the abundances for all intra-chromosomal bin pairs using the log-distance as the covariate. The bin size is added to the distance as a prior, to avoid undefined values upon log-transformation when distances are zero. The fitted value is then used as the threshold for each bin pair. For inter-chromosomal bin pairs, the threshold is set to that from the direct filtering approach, for completeness.

```
trended <- filterTrended(data)
```

The effect of this strategy can be visualized by plotting the interaction distance against the normalized abundance. A power-law relationship between distance and abundance is usually observed in Hi-C data [3]. The average abundance (and thus, the filter threshold) decreases as the distance between the interacting loci increases.

```
smoothScatter(trended$log.distance, trended$abundances,
              xlab="Log-Distance", ylab="Normalized abundance")
o <- order(trended$log.distance)
lines(trended$log.distance[o], trended$threshold[o], col="red", lwd=2)
```



The assumption here is that the majority of interactions are generated by non-specific packaging of the linear genome. Each bin pair is only retained if its abundance is greater than the corresponding fitted value at that distance, i.e., above that expected from compaction. This favors selection of longer-range interactions, compared to the direct filter.

```
trend.keep <- trended$abundances > trended$threshold
summary(trend.keep)

##    Mode   FALSE    TRUE
## logical 1445284 1873816
```

Of course, the threshold can also be defined at some minimum fold change above the fitted value. This effectively increases the stringency of the filter. The example below retains bin pairs with abundances that are two-fold higher than the expected compaction intensity.

```
trend.keep2 <- trended$abundances > trended$threshold + log2(2)
summary(trend.keep2)

##    Mode   FALSE    TRUE
## logical 3081407  237693
```

The trended filter can also be combined with `count.keep` to ensure that the absolute counts are large. This is particularly important at large distances, where the drop in the threshold may lead to the inappropriate retention of very low abundance bins.

```
trend.keep3 <- trend.keep & count.keep
```

The distance between bins can also be obtained directly with the `pairdist` function. Distances for inter-chromosomal bin pairs are marked as `NA`. These tend to dominate the output as they constitute most of the interaction space. Of course, the majority of these bin pairs will have low counts due to the sparseness of the data.

4.3.2 Using larger bin pairs to save memory

The procedure above assumes that no pre-filtering was performed on the counts for small bin pairs. In situations with limited memory, the counts for larger bin pairs can again be used to define the thresholds. The trend is fitted to the abundances and distances of the larger bin pairs, and interpolation (or extrapolation) is performed to obtain fitted values at the distances of the smaller bin pairs. The interpolated trend can then be applied to filter the smaller bin pairs based on their abundances. This entire process is done automatically within `filterTrended` when a `reference` argument is supplied, as shown below.

```
trended <- filterTrended(smaller.data, reference=data)
summary(trended$abundances > trended$threshold)

##    Mode   FALSE    TRUE
## logical  625765  435080
```

Another advantage of this approach is that threshold estimation is more precise with larger counts. Thus, even if there is enough memory for loading smaller bin pairs with `filter=1`, larger bin sizes may still be preferred for computing the filter threshold. Otherwise, median calculation or trend fitting will be confounded by discreteness at low counts.

4.4 Peak-calling in the interaction space

4.4.1 Defining enrichment values for each bin pair

Peaks in the interaction space are bin pairs that have substantially more reads than their neighbors. The `enrichedPairs` function counts the number of read pairs in the "neighborhood" of each bin pair. For a bin pair x, the function considers several neighborhood regions including bin pairs above or below x; in pairs to the left and right of x; bin pairs in a box centered at x; and for intra-chromosomal bin pairs, the quadrant closest to the diagonal [22]. The size of each neighborhood is determined by `flank`, defined in terms of bins. In this case, each bin is around 1 Mbp so the flank width has an actual size of ~5 Mbp.

```
flank.width <- 5
en.data <- enrichedPairs(data, flank=flank.width)
en.data

## class: InteractionSet
## dim: 3319100 4
## metadata(2): param width
## assays(5): counts quadrant vertical horizontal surrounding
## rownames: NULL
## rowData names(4): N.quadrant N.vertical N.horizontal N.surrounding
## colnames: NULL
## colData names(1): totals
## type: ReverseStrictGInteractions
## regions: 2739
```

The `filterPeaks` function uses the neighborhood counts to compute an enrichment value for each bin pair. The average abundance across libraries for each neighborhood region is computed, and the region with the largest abundance is chosen. The enrichment value is defined as the difference between the abundance of the bin pair and that of its chosen neighborhood region (adjusted for the number of bin pairs in the latter). Bin pairs with high enrichment values are putative peaks that should be retained. This extends the concept of peak calling on the linear genome (e.g., in ChIP-seq) to the 2-dimensional interaction space [22]. Neighborhood regions are defined to capture high-intensity structural features like looping domains, TADs or banding patterns. This ensures that the structural features themselves do not drive high enrichment values. Rather, to be called a peak, a bin pair in a structural feature must be enriched relative to other bin pairs in the same feature.

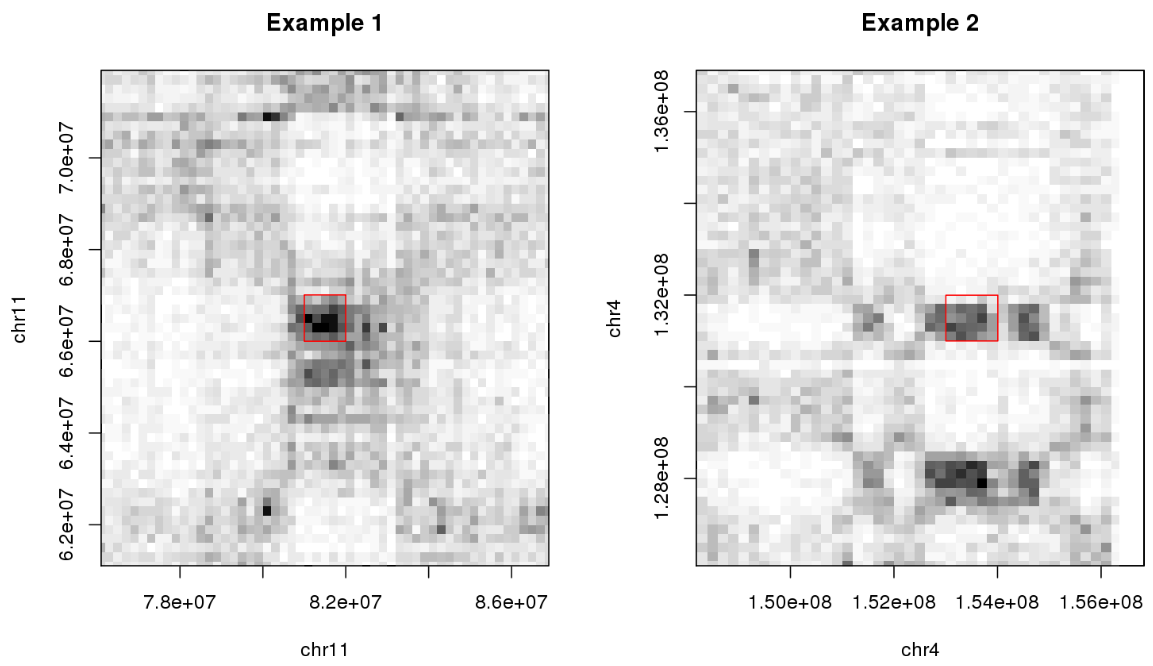
```
enrichments <- filterPeaks(en.data, get.enrich=TRUE)
summary(enrichments)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -11.57616  -0.46568  -0.17210       Inf   0.08292       Inf
```

In this example, an enrichment value above 0.5 is required for retention. This is a modest threshold that errs on the side of caution, as weak peaks may still be DIs and should be retained for testing. In practice, filtering of enrichment values should be combined with filtering on absolute abundances. This eliminates low-abundance bin pairs with high enrichment values, e.g., when the neighborhood is empty. Near-diagonal elements should also be removed, as these tend to have high enrichment scores without actually being peaks. All of these steps can be conveniently applied through the `filterPeaks` wrapper function.

```
peak.keep <- filterPeaks(data, enrichments, min.enrich=0.5,
                         min.count=5, min.diag=2L)
sum(peak.keep)

## [1] 88356
```

4.4.2 Examining some peak-calling diagnostics

This filtering strategy can be evaluated by examining the interaction space around each putative peak (see Chapter 7.5). Briefly, the color intensity of each "pixel" is proportional to the number of read pairs between the corresponding loci on each axis. Red boxes mark the bin pairs retained by filtering, where each side represents a bin. In both examples, the bin pairs correspond nicely to punctate high-intensity contacts in the interaction space.



Another diagnostic is based on the sparsity of putative peaks. Bin pairs nested within larger "parent" bin pairs are identified using the `boxPairs` function. Most of these parents should contain low numbers of nested bin pairs. This is because each peak, by definition, should be isolated in its neighborhood. The example below considers parent bin pairs of size equal to the neighborhood used to compute the enrichment values, and calculates the percentage of parents that contain a given number (from 1 to 10, or higher) of nested bin pairs.

```
neighborhood <- (2*flank.width + 1) * metadata(data)$width
boxed <- boxPairs(data[peak.keep], reference=neighborhood)
out <- tabulate(tabulate(boxed$indices[[1]]))
out <- c(out[1:10], sum(out[11:length(out)])) # sum all >10
setNames(round(out/sum(out)*100, 1), c(1:10, ">10"))

##    1    2    3    4    5    6    7    8    9   10  >10
## 22.3 22.3 18.2 13.0  8.8  5.6  3.5  2.3  1.6  0.9  1.5
```

Most parents should contain few bin pairs (≤ 10, as a rule of thumb), as shown above. In the second example below, a larger proportion of parents with many nested bin pairs is observed. This suggests that more aggressive filtering on the enrichment score is required.

```
peak.keep2 <- filterPeaks(data, enrichments, min.enrich=0,
                          min.count=5, min.diag=2L)
boxed <- boxPairs(data[peak.keep2], reference=neighborhood)
out <- tabulate(tabulate(boxed$indices[[1]]))
out <- c(out[1:10], sum(out[11:length(out)])) # sum all >10
setNames(round(out/sum(out)*100, 1), c(1:10, ">10"))

##    1    2    3    4    5    6    7    8    9   10  >10
##  5.4  4.3  4.0  3.7  3.7  3.4  3.1  2.9  2.8  2.7 64.0
```

Obviously, peak calling assumes that changes in intensity of broad interactions are not interesting. This may not be appropriate for every study. Indeed, the definition of "broad" depends on the bin size, whereby a peak called at large bin sizes may be discarded at smaller sizes. Users should try out the other, more general filters before proceeding to peak calling, to check that potentially interesting differences are not discarded by the latter.

4.4.3 Efficient peak calling at high resolution

The `enrichedPairs` function requires that all bin pairs are loaded into memory, i.e., with `filter=1` during `squareCounts`. This may not be feasible for high resolution analyses with small bin sizes, where there is a large number of bin pairs with low counts. In such cases, it may be more practical to use the `neighborCounts` function. This re-counts the read pairs for each bin pair, storing only a limited portion of the interaction space to compute the neighbourhood counts. The need to load all non-empty bin pairs is avoided. Only bin pairs with count sums greater than or equal to `filter` are returned, along with neighborhood counts that can be used in `filterPeaks`. This reduces memory usage at the cost of some speed.

```
en.data <- neighborCounts(input, param, width=1e5, filter=20, flank=5)
en.data

## class: InteractionSet
## dim: 1060845 4
## metadata(2): param width
## assays(5): counts quadrant vertical horizontal surrounding
## rownames: NULL
## rowData names(4): N.quadrant N.vertical N.horizontal N.surrounding
## colnames: NULL
## colData names(1): totals
## type: ReverseStrictGInteractions
## regions: 26729

enrichments <- filterPeaks(en.data, get.enrich=TRUE)
summary(enrichments)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -5.98142 -1.20472 -0.36342      Inf  0.08444      Inf
```

4.5 Filtering for pre-specified regions

Filtering for pairs of arbitrary regions is complicated by the potential irregularity of the regions. In particular, there is no guarantee that the supplied regions will cover the entirety of the interaction space. Filter thresholds may not be estimated accurately if the covered area is not representative of the rest of the space. At worst, genuine interactions between all specified regions would preclude direct or trended filtering, which assume that most interactions are uninteresting. That said, threshold estimation from a subset of the interaction space may actually be desirable in some cases, e.g., using only the captured regions to compute a relevant estimate of the non-specific ligation rate in a Capture-C experiment.

Another consideration is that different regions will have different widths. The resulting areas used for read pair counting will probably be different between region pairs, such that some will have systematically higher counts than others. Whether or not this is a problem depends on the user's perspective. The position of this guide is that it would be inappropriate to penalize larger areas for having larger counts. As long as there are sufficient counts for an analysis, the size of the area involved should be irrelevant. Thus, use of `aveLogCPM` alone is recommended for calculation of the average abundance when filtering region pairs.

```
summary(aveLogCPM(asDGEList(redata)))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -4.942  -4.939  -4.936  -4.753  -4.786  13.310
```

4.6 Filtering out diagonal elements

Incomplete removal of artifacts (e.g., dangling ends, self-circles) will generally manifest as counts for diagonal bin pairs, i.e., pairs of the same bin. If artifact generation and/or removal is not consistent between libraries, the behavior of the diagonal elements will differ markedly from the other bin pairs. This can be diagnosed using MA plots between libraries, where the diagonal elements show up as a distinct mass that is unconnected to the other points. As such, they cannot be smoothly normalized and should be removed prior to further analysis or analyzed separately. This is done by generating `ndiag.keep` for filtering, as shown below.

```
ndiag.keep <- filterDiag(data)
summary(ndiag.keep)

##    Mode   FALSE    TRUE
## logical    2614 3316486
```

Removal of diagonal elements is also motivated by the difficulty of intepreting interactions within a bin. For small bin sizes, read pairs corresponding to internal interactions are indistinguishable from those driven by non-specific packaging of a locus. The latter is mostly uninteresting and can be avoided by discarding the bin pairs on the diagonal. For large bin sizes, internal interactions may be more interesting – these can be recovered by repeating the analysis with smaller bins. Short-range interactions will then be represented by the off-diagonal pairs, while artifacts will still be restricted to diagonal bin pairs, so long as the bins are larger than ~25 kbp (see Section 2.4.2). Count sizes will also decrease but this should not be a major problem as counts should be inherently larger at low interaction distances.

4.7 Summary of the filtering strategies

Each filtering strategy can be tuned by increasing or decreasing the minimum fold change required for retention. This can be driven by the biological knowledge available to the user, based on features that are biologically interesting. Even when such knowledge is unavailable, the filters are still useful as they can guide the user towards a sensible interpretation of the filter threshold. This would not be possible if the threshold value was chosen arbitrarily.

The choice of filtering method depends on the features that are most likely to be of interest in each analysis. In general, less aggressive filtering should be performed if these features are not well defined. Here, a tentative recommendation is provided for direct filtering as it is reasonable to discard non-specific ligation events. On a practical level, the simplicity of the direct approach is attractive and will be used throughout the rest of the guide.

```
original <- data
data <- data[direct.keep2,]
```

The filtered results are assigned back into `data`. This is mostly for consistency, so that all operations are performed on `data` in the rest of this guide. The original unfiltered data is retained for later use in Section 5.3, but can otherwise be ignored. Direct filtering is also performed for the smaller bin pairs, and the results stored in `smaller.data`.

```
smaller.data <- smaller.data[small.keep,]
```

Chapter 5

Normalization strategies for Hi-C data

Here, we're using the `data` object that was filtered in the previous chapter. We'll also need the `original` object, as well as `margin.data` from Section 3.4. Another human dataset will be loaded here, so `hs.frag` from Chapter 2 will be required.

5.1 Normalizing with scaling methods

The simplest approach is to use scaling methods such as library size normalization. This accounts for differences in library preparation efficiency and sequencing depth between samples. In *edgeR*, scaling is performed using the effective library size, defined as the product of the library size and the normalization factor for each sample (see the *edgeR* user's guide). Library size normalization is thus achieved by setting all normalization factors to unity.

```
data.lib <- data # Making a copy to avoid side-effects.
data.lib$norm.factors <- 1
```

Alternatively, TMM normalization [23] can be applied on the counts for the bin pairs. This accounts for composition biases by assuming that most interactions do not change in intensity between conditions. Here, the normalization factors differ from unity as the library size alone is not sufficient to describe the bias in each sample.

```
data.tmm <- normOffsets(data, se.out=TRUE)
data.tmm$norm.factors

## [1] 0.8953459 0.8979718 1.1127467 1.1177640
```

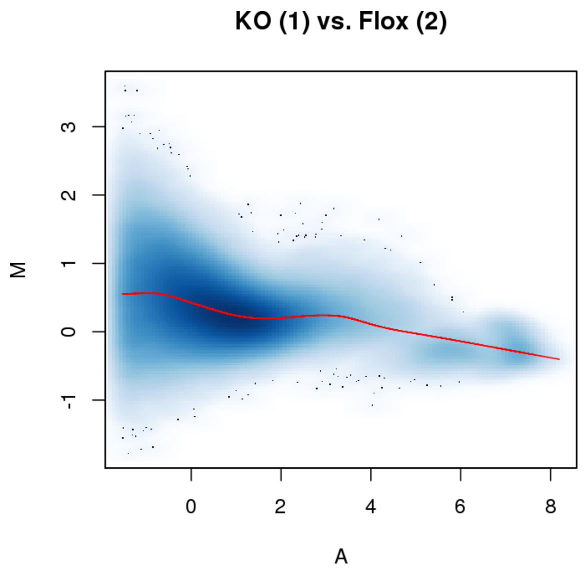
In practice, scaling methods are usually too simplistic. More sophisticated approaches are necessary to handle the complex biases observed in real Hi-C data.

5.2 Removing trended biases between libraries

5.2.1 Using non-linear normalization

Trended biases may be observed in Hi-C data, caused by uncontrolled differences in sample preparation in a complex protocol. Changes in cross-linking efficiency or ligation specificity can lead to a systematic redistribution of read pairs throughout the interaction space. For example, reduced specificity may result in more counts for weak non-specific interactions, and fewer counts for strong genuine interactions. Such biases may manifest as an abundance-dependent trend in a MA plot between libraries. This is especially true for a trend observed between replicates, which must be technical in origin. The code below generates a MA plot comparing one library from each group, which can be repeated for each pair of libraries.

```
ab <- aveLogCPM(asDGEList(data))
o <- order(ab)
adj.counts <- cpm(asDGEList(data), log=TRUE)
mval <- adj.counts[,3]-adj.counts[,2]
smoothScatter(ab, mval, xlab="A", ylab="M", main="KO (1) vs. Flox (2)")
fit <- loessFit(x=ab, y=mval)
lines(ab[o], fit$fitted[o], col="red")
```



Trended biases are problematic as they can inflate the variance estimates or fold-changes for some bin pairs. They must be eliminated with non-linear normalization prior to further analysis. We use LOESS normalization with the `normOffsets` function from the *csaw* package [24], adapted from existing non-linear methods to handle discrete count data.

```
data <- normOffsets(data, type="loess", se.out=TRUE)
```

This function computes an offset term for each bin pair in each library. When fitting a generalized linear model (GLM) to the counts for a particular bin pair, a large offset for a library is equivalent to downscaling the corresponding count relative to the counts of other libraries. The matrix of offsets has the same dimensions as the count matrix and is stored as an element of the `assays` slot of the *InteractionSet* object. (Specifying `se.out=FALSE` returns the offset matrix directly, instead of a modified *InteractionSet* containing the offsets.)

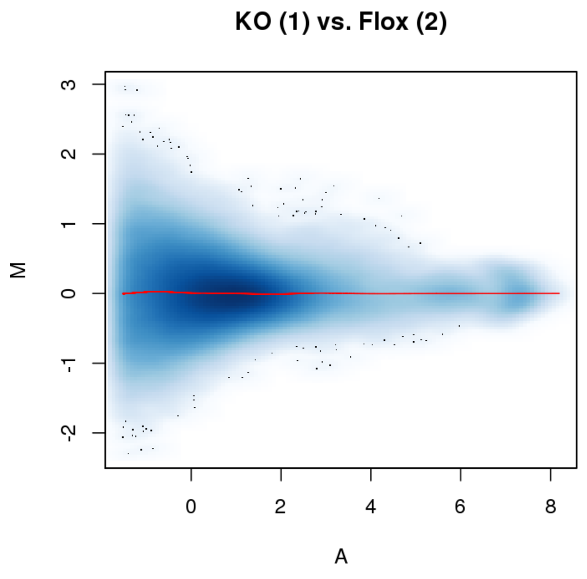
```
nb.off <- assay(data, "offset")
head(nb.off)

##            [,1]       [,2]       [,3]        [,4]
## [1,] 0.1359035 -0.1111409  0.0520197 -0.07678226
## [2,] 0.3827167  0.2058267 -0.2489948 -0.33954867
## [3,] 0.3021307  0.1516470 -0.1829528 -0.27082491
## [4,] 0.3536789  0.1864192 -0.2252649 -0.31483322
## [5,] 0.2285073  0.1019900 -0.1225945 -0.20790279
## [6,] 0.2932061  0.1455843 -0.1756060 -0.26318443
```

In most cases, the above code is sufficient for normalization. However, as previously mentioned, bin pairs near the diagonal may exhibit irregular behavior relative to the rest of the interaction space. This manifests in the previous MA plot as a distinct mass of points at high abundances, preventing smooth normalization of the trended bias. To overcome this, the near-diagonal bin pairs can be normalized separately from the others.

```
neardiag <- filterDiag(data, by.dist=1.5e6)
nb.off <- matrix(0, nrow=nrow(data), ncol=ncol(data))
nb.off[neardiag] <- normOffsets(data[neardiag,], type="loess", se.out=FALSE)
nb.off[!neardiag] <- normOffsets(data[!neardiag,], type="loess", se.out=FALSE)
assay(data, "offset") <- nb.off # updating the offset matrix
```

We examine the MA plot after adjusting the log-counts with the offsets. Most of the trend is removed, indicating that non-linear normalization was successful.

```
adj.counts <- log2(assay(data) + 0.5) - nb.off/log(2)
mval <- adj.counts[,3]-adj.counts[,2]
smoothScatter(ab, mval, xlab="A", ylab="M", main="KO (1) vs. Flox (2)")
fit <- loessFit(x=ab, y=mval)
lines(ab[o], fit$fitted[o], col="red")
```



5.2.2 Requirements of non-linear normalization

Filtering on average abundance prior to normalization is strongly recommended. This removes low counts and avoids problems with discreteness during trend fitting. Filtering also improves the sensitivity of span-based fitting algorithms like LOESS at higher abundances. Otherwise, the fitted trend will be dominated by the majority of low-abundance bin pairs.

Non-linear normalization also assumes that the majority of bin pairs at each abundance do not represent differential interactions. Any systematic differences between libraries are assumed to be technical in origin and are removed. If this assumption does not hold, genuine differences may be lost upon normalization. For example, a global increase in the compaction of the genome would manifest as an upward trend in the MA plot, as low-abundance distal interactions are weakened in favor of high-abundance short-range interactions. In such cases, use of library size normalization in Section 5.1 may be more appropriate.

5.3 Iterative correction of interaction intensities

While this is not the intended purpose of the *diffHic* package, a method is also provided for the removal of biases between genomic regions. The `correctedContact` function performs iterative correction of the interaction space [6] with some modifications. Namely, if multiple libraries are used to present, correction is performed using the overall NB mean for each bin pair, rather than on the counts themselves. Winsorizing through `winsor.high` is also performed to mitigate the impact of high-abundance bin pairs.

```
corrected <- correctedContact(original, winsor.high=0.02, ignore.low=0.02)
head(corrected$truth)

## [1] 0.45646178 0.43318032 0.14752258 0.15233067 0.26725581 0.03447091
```

The returned `truth` contains the "true" contact probability for each bin pair in `data`. This is designed to account for differences in sequencibility, mappability, restriction site frequency, etc. between bins. Comparisons can then be directly performed between the contact probabilities of different bin pairs. Some `NA` values will be present due to the removal of low-abundance bins that do not exhibit stable behavior during correction. The convergence of the correction procedure can then be checked by examining the maximum fold change to the truth at each iteration. This should approach unity, i.e., no further change.

```
corrected$max

##  [1] 186.415803   8.018365   2.848421   1.703012   1.324769   1.170798
##  [7]   1.098717   1.060892   1.039169   1.025855   1.017337   1.011736
## [13]   1.007991   1.005461   1.003740   1.002565   1.001761   1.001209
## [19]   1.000831   1.000571   1.000393   1.000270   1.000185   1.000127
## [25]   1.000088   1.000060   1.000041   1.000028   1.000020   1.000013
## [31]   1.000009   1.000006   1.000004   1.000003   1.000002   1.000002
## [37]   1.000001   1.000001   1.000001   1.000000   1.000000   1.000000
## [43]   1.000000   1.000000   1.000000   1.000000   1.000000   1.000000
## [49]   1.000000   1.000000
```

Note that `original` is used as the input to `correctedContact`. No filtering should be performed prior to iterative correction. All non-empty bin pairs are needed as information is collected across the entire interaction space. The contribution of many bin pairs with low counts might end up being as substantial as that of a few bin pairs with large counts.

In addition, normalization from iterative correction can be fed naturally into the DI analysis via the GLM machinery in *edgeR*. The log-transformed product of the estimated genomic biases for both bins in each bin pair can be used as the offset for that bin pair. This is computed separately for each library by setting `average=FALSE`, to correct for sample-specific genomic biases. `NA` offsets will be obtained for some bin pairs, but these should not be too problematic given that the affected bin pairs will generally have low counts (as at least one interacting partner will be of low abundance) and be filtered out during the analysis proper.

```
corrected <- correctedContact(original, average=FALSE)
anchor1.bias <- corrected$bias[anchors(original, type="first", id=TRUE),]
anchor2.bias <- corrected$bias[anchors(original, type="second", id=TRUE),]
iter.off <- log(anchor1.bias * anchor2.bias)
```

Of course, iterative correction only removes biases between different bins. It is not guaranteed to remove (trended) biases between libraries. For example, two replicates could have the same genomic biases but a different distribution of read pairs in the interaction space, e.g., due to differences in ligation specificity. The latter would result in a trended bias, but iterative correction would have no effect due to the identical genomic biases. In short, normalization within a library is a different problem from normalization between libraries.

5.4 Accounting for copy number variations

5.4.1 Eliminating CNVs with multi-dimensional smoothing

Copy number variations (CNVs) in the interacting regions will also affect the interaction intensity. These CNV-driven differences in intensities are generally uninteresting and must be removed to avoid spurious detection of DIs. Normalization of CNV-based biases is achieved using the `normalizeCNV` function, which performs multi-dimensional smoothing across several covariates with the *locfit* package [25]. It requires both bin pair and marginal counts, obtained with the same `width` and `param` in calls to `squareCounts` and `marginCounts`.

```
cnv.offs <- normalizeCNV(data, margin.data)
head(cnv.offs)

##           [,1]        [,2]        [,3]       [,4]
## [1,] 0.3849393 -0.24953617  0.05145434 -0.1868574
## [2,] 0.3883340  0.15601877 -0.26384559 -0.2805072
## [3,] 0.2990999  0.13624039 -0.20061591 -0.2347244
## [4,] 0.3509612  0.17525088 -0.24302170 -0.2831904
## [5,] 0.2285712  0.08363148 -0.13948501 -0.1727176
## [6,] 0.2925324  0.13509101 -0.19407793 -0.2335454
```

Three covariates are defined for each bin pair. For a pair of libraries, the ratio of marginal counts for each bin can be used as a proxy for the relative CNV between libraries in that bin. Each bin pair will be associated with two of these marginal log-ratios to use as covariates. The third covariate for each bin pair is that of the average abundance across all libraries. This will account for any abundance-dependent trends in the biases.

The response for each bin pair is defined as the log-ratio of interaction counts between a pair of libraries. A locally weighted surface is fitted to the response against all three covariates for all bin pairs. At any combination of covariate values, most bin pairs are assumed to represent non-differential interactions. Any systematic differences between libraries are attributed to CNV-driven (or trended) biases and are removed. Specifically, GLM offsets are returned and can be supplied to the statistical analysis to eliminate the bias.

As with trended biases, filtering by average abundance is strongly recommended prior to running `normalizeCNV`. This reduces the computational work required for multi-dimensional smoothing. Discreteness and domination of the fit by low-abundance bin pairs is also avoided.

5.4.2 Visualizing the effect of CNV removal

To demonstrate, the RWPE1 dataset [5] is used here as it contains more CNVs. Interaction counts are loaded for each bin pair, and marginal counts are loaded for each bin. Some filtering is performed to eliminate low-abundance bin pairs, as previously described.

```
count.files <- c("merged_erg.h5", "merged_gfp.h5")
rick.data <- squareCounts(count.files, hs.param, width=1e6)
rick.marg <- marginCounts(count.files, hs.param, width=1e6)
rick.data <- rick.data[aveLogCPM(asDGEList(rick.data)) > 0,]
```

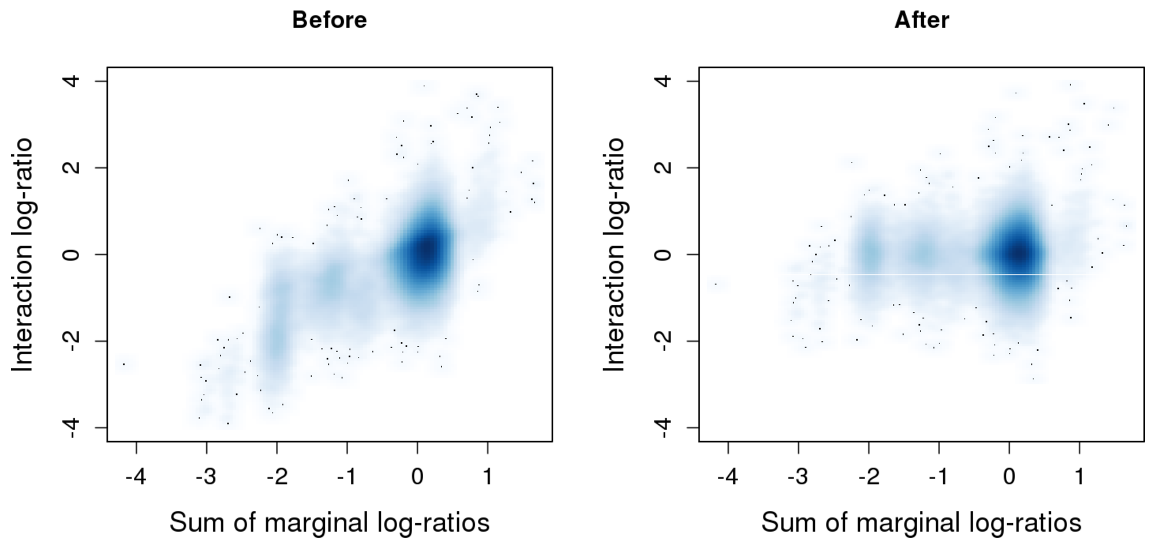
Our aim is to plot the log-ratio of the counts for each bin pair between the first two libraries, as a function of the log-ratio of the marginal counts of the corresponding bins. The code below computes the marginal log-ratios after matching the bins in `rick.marg` to the bin pairs in `rick.data` with `matchMargins`. As two marginal log-ratios are present for each bin pair, these are added together to simplify visualization.

```
m.adjc <- cpm(asDGEList(rick.marg), log=TRUE)
margin.lr <- m.adjc[,1] - m.adjc[,2]
matched <- matchMargins(rick.data, rick.marg)
margin.lr <- margin.lr[matched$anchor1] + margin.lr[matched$anchor2]
```

We generate plots before and after subtracting the offsets computed by `normalizeCNV`. Before normalization, we observe that a decrease in copy number results in a decrease in interaction intensity. This is expected given that fewer copies should result in fewer chances for interaction. After normalization, the trend in the interaction log-ratios is removed. This indicates that the CNV effect has been eliminated and can be ignored in downstream analyses. Note that increasing `maxk` may be necessary to obtain accurate results in the internal call to *locfit*.

```
before <- cpm(asDGEList(rick.data), log=TRUE)
cnv.offs <- normalizeCNV(rick.data, rick.marg, maxk=1000)
after <- log2(assay(rick.data)+0.5) - cnv.offs/log(2)
par(mfrow=c(1,2), cex.axis=1.2, cex.lab=1.4)
smoothScatter(margin.lr, before[,1]-before[,2], ylim=c(-4, 4), main="Before",
    xlab="Sum of marginal log-ratios", ylab="Interaction log-ratio")
```

```
smoothScatter(margin.lr, after[,1]-after[,2], ylim=c(-4, 4), main="After",
    xlab="Sum of marginal log-ratios", ylab="Interaction log-ratio")
```



5.4.3 Using alternative copy number definitions

The method above uses the marginal counts as a proxy for genomic coverage [6]. Changes in the marginal counts can be used as a proxy for change in the copy number between samples. In most cases, this is an effective and sufficient strategy as exact quantification of the copy number difference is not required. However, external information can also be used, based on copy number arrays or genomic sequencing data for each sample. The (relative) copy number for each bin/region in each sample can be stored in a *RangedSummarizedExperiment* and passed to `normalizeCNV` as shown above. This may be more accurate as it avoids potential problems from interaction-specific effects on the marginal counts.

Chapter 6

Modelling biological variability

In this chapter, the `data` object is again required. The computed offsets in `nb.off` will also be used from the last chapter. Methods from the *edgeR* package should already be loaded into the workspace, but if they aren't, then `library(edgeR)` will do the job.

6.1 Overview

The differential analysis in *diffHic* is based on the statistical framework in the *edgeR* package [1]. This models the counts for each bin pair with NB distributions. The NB model is useful as it can naturally accommodate low, discrete counts. It can also consider extra-Poisson variability between biological replicates of the same condition. Here, biological replicates refer to Hi-C libraries prepared from independent biological samples.

The magnitude of biological variability is empirically determined from these biological replicates. In *edgeR*, variability is modelled by estimating the dispersion parameter of the NB distribution. This is used during testing to reduce the significance of any detected differences when the counts are highly variable. Similarly, estimation of the quasi-likelihood (QL) dispersion can be performed to model variability of the dispersions [26].

Dispersion estimation requires the fitting of a GLM to the counts for each bin pair [19]. To do so, a design matrix must be specified to describe the experimental setup. For the neural stem cell dataset, a simple one-way layout is sufficient. The code below specifies two groups of two replicates, where each group corresponds to a genotype. The aim is to compute the dispersion from the variability in counts within each group.

```
design <- model.matrix(~factor(c("flox", "flox", "ko", "ko")))
colnames(design) <- c("Intercept", "KO")
design

##   Intercept KO
## 1         1  0
## 2         1  0
## 3         1  1
## 4         1  1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$`factor(c("flox", "flox", "ko", "ko"))`
## [1] "contr.treatment"
```

Obviously, more complex designs can be used if necessary. This includes designs with blocking factors for batch effects, pairing between samples or multiple groups.

The *InteractionSet* object also needs to be converted into a *DGEList* object for analysis with *edgeR*. If the normalization factors or offsets are already stored in `data`, as previously described, then they will be automatically extracted by the `asDGEList` function and stored in the output *DGEList*. Alternatively, they can be passed explicitly to `asDGEList`. Note that the offset matrix will take precedence over scaling factors in *edgeR* analyses.

```
y <- asDGEList(data)
```

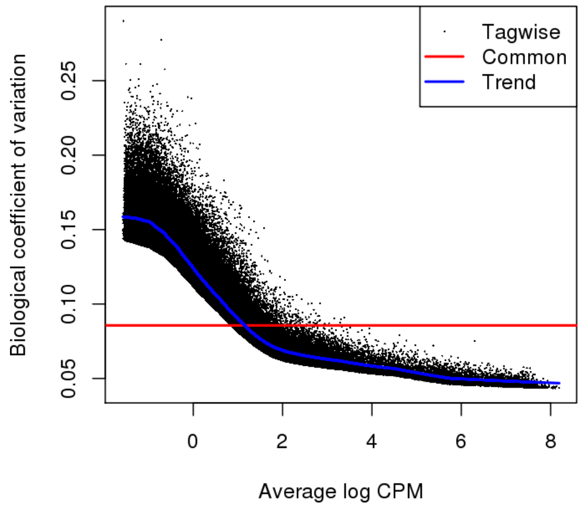
6.2 Estimating the NB dispersion

Estimation of the NB dispersion is performed by maximizing the Cox-Reid adjusted profile likelihood (APL) [19] for each bin pair. Of course, when replication is limited, there is not enough information per bin pair to estimate the dispersion. This is overcome by computing and sharing APLs across many bin pairs to stablize the estimates.

```
y <- estimateDisp(y, design)
y$common.dispersion

## [1] 0.007331746
```

A more sophisticated strategy is also used whereby an abundance-dependent trend is fitted to the APLs. This should manifest as a smooth trend in the NB dispersion estimates with respect to the average abundances of all bin pairs. The aim is to improve modelling accuracy by empirically modelling any non-NB mean-variance relationships.

```
plotBCV(y)
```



In most cases, the relationship should be monotonic decreasing as the counts become more precise with increasing size. Minor deviations are probably due to the imperfect nature of non-linear normalization. Major increases are indicative of batch effects. For example, a cluster of outliers indicates that there may be copy number changes between replicates.

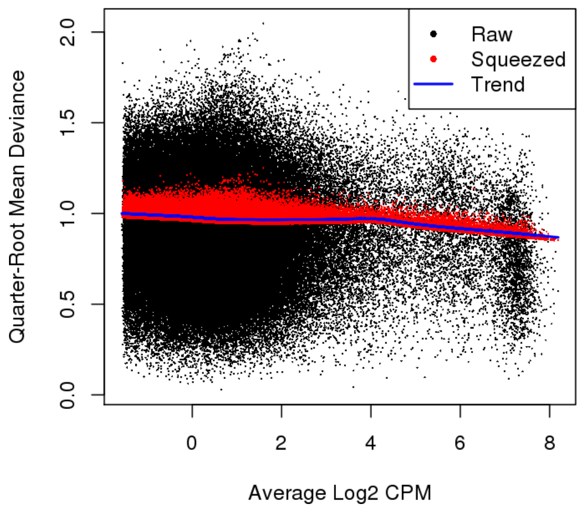
6.3 Estimating the QL dispersion

The QL dispersion for each bin pair is estimated from the deviance of the fitted GLM. This may seem superfluous given that the NB dispersion already accounts for biological variability. However, it is mathematically easier to model variability in the QL dispersions across bin pairs, compared to variability in the NB dispersions. GLM fitting and estimation of the QL dispersion for each bin pair are performed with the `glmQLFit` function.

```
fit <- glmQLFit(y, design, robust=TRUE)
```

Again, there is not enough information for each bin pair for precise estimation. Instead, information is shared between bin pairs using an empirical Bayes (EB) approach. Per-bin-pair QL estimates are shrunk towards a common trend across all bin pairs. This stabilizes the QL dispersion estimates and improves precision for downstream applications.

```
plotQLDisp(fit)
```



The extent of the EB shrinkage is determined by the variability of the dispersions. If the true dispersions are highly variable, shrinkage to a common value would be inappropriate. On the other hand, more shrinkage can be performed to increase precision (and thus detection power) if the true dispersions are not variable. This is quantified as the prior degrees of freedom, for which smaller values correspond to more variability and less shrinkage.

```
summary(fit$df.prior)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   19.12   19.82   19.82   19.82   19.82   19.82
```

It is important to use the `robust=TRUE` argument in `glmQLFit` [27]. This protects the EB shrinkage against large positive outliers corresponding to highly variable counts. It also protects against large negative outliers. These are formed from near-zero deviances when counts are identical, and are not uncommon when counts are low. Both types of outliers inflate the apparent variability and decrease the estimated prior degrees of freedom.

6.4 Further information

More details on the statistical methods in *edgeR* can be found, unsurprisingly, in the *edgeR* user's guide. Of course, *diffHic* is compatible with any statistical framework that accepts a count matrix and a matrix of log-link GLM offsets. Advanced users may wish to use methods from other packages. This author prefers *edgeR* as it works quite well for routine analyses of Hi-C data. He also has a chance of being cited when *edgeR* is involved [28].

Chapter 7

Testing for significant interactions

This chapter brings everything together. We need the `fit` object from the last chapter, along with the `data` object (as usual). We also require the `smaller.data` object from Chapter 4. The `bin.size` and `mm.param` objects are required from Chapter 3.

7.1 Using the quasi-likelihood F-test

The `glmQLFTest` function performs a QL F-test for each bin pair to identify significant differences. Users should ensure that the correct contrast is performed by explicitly specifying the `coef` or `contrast` arguments. In this case, the coefficient of interest refers to the change in the KO counts over the WT counts. The null hypothesis for each bin pair is that the coefficient is equal to zero, i.e., there is no change between the WT and KO groups.

```
result <- glmQLFTest(fit, coef=2)
topTags(result)

## Coefficient:  KO
##          logFC   logCPM        F       PValue          FDR
## 93599 1.383842 4.168789 230.8401 4.357777e-13 3.961340e-08
## 84768 1.540142 3.157513 227.8982 4.954369e-13 3.961340e-08
## 84766 1.540032 2.405414 166.1560 1.114922e-11 5.434360e-07
## 2541  1.385410 2.977601 160.1627 1.590940e-11 5.434360e-07
## 58458 1.031685 4.673526 159.0746 1.699161e-11 5.434360e-07
## 84767 1.203459 3.313556 150.3301 2.926939e-11 7.800928e-07
## 93501 1.075536 3.844916 145.9064 3.895252e-11 8.898593e-07
## 84765 1.418004 2.290288 141.1296 5.349632e-11 1.047273e-06
## 59485 1.054100 4.004050 138.8398 6.249153e-11 1.047273e-06
## 2542  1.047167 4.093494 138.1557 6.549016e-11 1.047273e-06
```

More savvy users might wonder why the likelihood ratio test (LRT) was not used here. Indeed, the LRT is the more obvious test for any inferences involving GLMs. However, the QL F-test is preferred as it accounts for the variability and uncertainty of the QL dispersion estimates [26]. This means that it can maintain more accurate control of the type I error rate compared to the LRT in the presence of limited replication.

It is also convenient to store the significance statistics in the `rowData` of the *InteractionSet* object. This ensures that any operations like subsetting are applied on both the genomic coordinates and statistics simultaneously, which simplifies data management.

```
rowData(data) <- cbind(rowData(data), result$table)
```

7.2 Multiplicity correction and the FDR

7.2.1 Overview

In a *diffHic* analysis, many bin pairs are tested for significant differences across the interaction space. Correction for multiple testing is necessary to avoid excessive detection of spurious differences. For genome-wide analyses, this correction is typically performed by controlling the false discovery rate (FDR) with the Benjamini-Hochberg (BH) method [29]. This provides a suitable comprimise between specificity and sensitivity. In contrast, traditional methods of correction (e.g., Bonferroni) are often too conservative.

Interpretation of the differential interactions (DIs) is complicated by spatial dependencies between adjacent bin pairs. If many adjacent bin pairs were separately reported as DIs, this would complicate the interpretation of the results by introducing unnecessary redundancy. To mitigate this effect, *diffHic* provides several methods for clustering bin pairs and controlling the relevant FDR. Each of these methods is described below, along with its relative strengths and weaknesses. For routine analyses, the approach based on clustering significiant bin pairs is recommended – see Section 7.2.5 for details.

7.2.2 Direct application of the BH method

The simplest approach is to directly apply the BH method to the set of p-values for all tested bin pairs. The FDR here refers to the proportion of detected bin pairs that are false positives. DIs are defined as those bin pairs detected at a given FDR threshold, e.g., 5%. Their coordinates are saved to file along with the bin pair-based test statistics. Resorting on p-value is performed to prioritize strong DIs, thus simplifying inspection of the results.

```
adj.p <- p.adjust(result$table$PValue, method="BH")
sum(adj.p <= 0.05)

## [1] 4431

useful.cols <- as.vector(outer(c("seqnames", "start", "end"), 1:2, paste0))
inter.frame <- as.data.frame(interactions(data))[,useful.cols]
results.r <- data.frame(inter.frame, result$table, FDR=adj.p)
o.r <- order(results.r$PValue)
write.table(results.r[o.r,], file="binpairs.tsv", sep="\t",
            quote=FALSE, row.names=FALSE)
```

This approach is best suited for analyses using large bins, i.e., greater than or equal to 1 Mbp. Here, adjacent bin pairs need not be considered redundant as they cover distinct areas of the interaction space. Thus, no clustering is required to simplify interpretation.

7.2.3 Clustering to reduce redundancy in the results

Adjacent bin pairs can be aggregated into larger clusters to reduce redundancy in the results. This is especially useful at smaller bin sizes where multiple bin pairs may overlap a single underlying interaction. Each cluster of bin pairs will then represent the underlying interaction. This is demonstrated below with the 100 kbp bin pairs. The `clusterPairs` function puts two bin pairs in the same cluster if they are no more than `tol` bp apart in any dimension.

```
clustered.small <- clusterPairs(smaller.data, tol=1, upper=1e6)
```

In practice, this approach requires aggressive filtering to avoid chaining effects. Otherwise, clustering will be confounded in high-density areas of the interaction space, e.g., at short distances or in TADs. This leads to the formation of very large clusters that are difficult to interpret. Some protection is provided by specifying the maximum dimensions of each cluster in `upper`. This will break up very large clusters, albeit in a somewhat arbitrary manner.

There is also no guarantee that the cluster will form a regular shape in the interaction space. In `clusterPairs`, an approximate solution is used whereby the minimum bounding box for each cluster is reported. This refers to the smallest rectangle in the interaction space that contains all bin pairs in the cluster. The coordinates of this rectangle can be easily recorded, whereas it is more difficult to store the detailed shape of the cluster. Identification of the top-ranking bin pair within each cluster may also be desirable (see Section 7.4).

```
length(clustered.small$interactions)

## [1] 33139

head(clustered.small$interactions)

## ReverseStrictGInteractions object with 6 interactions and 0 metadata columns:
##       seqnames1             ranges1     seqnames2             ranges2
##           <Rle>           <IRanges>         <Rle>           <IRanges>
##   [1]      chr1 [3004106, 3100835] ---      chr1 [      1, 3004109]
##   [2]      chr1 [3004106, 4000741] ---      chr1 [3004106, 4000741]
##   [3]      chr1 [4000738, 5001375] ---      chr1 [3004106, 4000741]
##   [4]      chr1 [4000738, 5001375] ---      chr1 [4000738, 5001375]
##   [5]      chr1 [5001372, 5997485] ---      chr1 [3004106, 4000741]
##   [6]      chr1 [5001372, 5997485] ---      chr1 [4000738, 5001375]
##   -------
##   regions: 29896 ranges and 0 metadata columns
##   seqinfo: 66 sequences from an unspecified genome
```

The FDR across clusters is not the same as that across bin pairs. The former is more useful as the results are usually interpreted in terms of clusters, where each cluster corresponds to one interaction event. However, the FDR across bin pairs is easier to control by applying the BH method directly to the bin pair p-values. Treating the FDR across bin pairs as that across clusters is usually inappropriate and results in loss of FDR control [30]. The following text will describe some strategies for controlling the cluster-level FDR.

7.2.4 Independent clustering of adjacent bin pairs

The basic idea of independent clustering is that it is done independently of the differential test statistics. This means that other metrics must be used to identify relevant bin pairs. If there are pre-defined areas of interest in the interaction space (e.g., based on existing interactions from earlier experiments or when studying interactions between annotated features), identification of these bin pairs can be performed with methods like `findOverlaps` from the *InteractionSet* package. All bin pairs overlapping a pre-defined area can then be defined as a single cluster. Otherwise, filtering on the average abundance will implicitly restrict the clustering to high-abundance bin pairs. This may be satisfactory when the interaction space is sparse such that the clusters will be small enough to be interpretable. To demonstrate, a quick-and-dirty differential analysis is performed using counts for the 100 kbp bin pairs.

```
smaller.data <- normOffsets(smaller.data, type="loess", se.out=TRUE)
y.small <- asDGEList(smaller.data)
y.small <- estimateDisp(y.small, design)
fit.small <- glmQLFit(y.small, design, robust=TRUE)
result.small <- glmQLFTest(fit.small)
```

A combined p-value is computed for each cluster from the p-values of its bin pairs, using Simes' method [31] as implemented in the `combineTests` function from the *csaw* package. Each combined p-value represents the evidence against the global null hypothesis, i.e., that none of the constituent bin pairs are significant in the cluster. Cluster-level FDR control is maintained by applying the BH method on the combined p-values for all clusters. Significant clusters are then defined at a certain FDR threshold, e.g., 5%.

```
library(csaw)
tabcluster <- combineTests(clustered.small$indices[[1]], result.small$table)
head(tabcluster)

##   nWindows logFC.up logFC.down       PValue          FDR direction
## 1        1        0          1 3.251461e-02 1.388666e-01      down
## 2       55        0          0 2.636977e-02 1.190695e-01        up
## 3       99        1         10 2.925148e-01 5.475090e-01     mixed
## 4       55        0          6 8.591867e-10 7.521332e-08      down
## 5       99       26          4 1.598432e-01 3.888596e-01        up
## 6      100        1         14 8.480900e-05 1.027224e-03      down

sum(tabcluster$FDR <= 0.05)

## [1] 5629
```

The `combineTests` function also reports details about each cluster, including the total number of bin pairs and the number changing in each direction. These results are saved to file below, along with the coordinates of the minimum bounding box for each cluster.

```
inter.frame <- as.data.frame(clustered.small$interactions)[,useful.cols]
results.i <- data.frame(inter.frame, tabcluster)
o.i <- order(results.i$PValue)
write.table(results.i[o.i,], file="independent.tsv", sep="\t",
            quote=FALSE, row.names=FALSE)
```

The statistics can also stored in the metadata of the *GInteractions* object returned by `clusterPairs`. This ensures that the genomic coordinates and statistics are synchronised.

```
mcols(clustered.small$interactions) <- cbind(
    mcols(clustered.small$interactions), tabcluster)
```

7.2.5 Clustering based on significant bin pairs

The interaction space is often dominated by high-abundance structural features like domains and compartments. In such high-density areas, too many bin pairs may be retained after independent filtering on abundance. Clustering would then result in excessive chaining, yielding very large clusters for which the coordinates provide no information on DIs.

To avoid this, it may be necessary to cluster using only those bin pairs that are significantly different. These bin pairs are more sparsely distributed throughout the interaction space, such that the coordinates of the resulting clusters can be easily interpreted. This is done using the `diClusters` function, as shown below for the results with the 1 Mbp bin pairs. Each cluster represents a DI containing only significant bin pairs. This is more useful than reporting the coordinates for an entire domain if only a portion of the domain is changing.

```
clustered.sig <- diClusters(data, result$table, target=0.05,
                            cluster.args=list(tol=1))
length(clustered.sig$interactions)

## [1] 4430

head(clustered.sig$interactions)

## ReverseStrictGInteractions object with 6 interactions and 0 metadata columns:
##       seqnames1              ranges1     seqnames2             ranges2
##           <Rle>            <IRanges>         <Rle>           <IRanges>
##   [1]      chr1 [10000179, 11003377] ---      chr1 [8999921, 10000182]
##   [2]      chr1 [11003374, 11998773] ---      chr1 [7000257,  8000015]
##   [3]      chr1 [11003374, 11998773] ---      chr1 [8000012,  8999924]
##   [4]      chr1 [11003374, 11998773] ---      chr1 [8999921, 10000182]
##   [5]      chr1 [11998770, 12998158] ---      chr1 [5997482,  7000260]
##   [6]      chr1 [11998770, 12998158] ---      chr1 [7000257,  8000015]
##   -------
##   regions: 2195 ranges and 0 metadata columns
##   seqinfo: 66 sequences from an unspecified genome
```

The `clusterPairs` function is used internally to cluster significant bin pairs. No limits are placed on the maximum dimensions as the sparsity of the selected bin pairs should avoid chaining effects. Additional sophistication can be obtained by setting `fc.col="logFC"` above. This will cluster bin pairs separately if they are changing in opposite directions.

The `diClusters` function will attempt to control the cluster-level FDR below `target`, i.e., 5%. Specifically, the cluster-level FDR is estimated from the FDR threshold used to define significant bin pairs (see the `clusterFDR` function in the *csaw* package). The bin pair-level threshold is then adjusted until the estimate of the cluster-level FDR lies close to `target`. The estimated cluster-level FDR is stored in the `FDR` field of the output. Formally speaking, this procedure is not entirely rigorous and will yield biased estimates for small numbers of clusters. However, this may be a necessary price to pay for interpretable results.

```
clustered.sig$FDR

## [1] 0.04988713
```

The identities of the bin pairs in each cluster are also returned in the `indices` field of the output list. These can be used to compute additional statistics for each cluster using the `combineTests` and `getBestTest` functions in *csaw*. The latter will identify the bin pair with the lowest p-value, which is useful for finding the strongest changes in large clusters. Users are advised to ignore the p-value and FDR fields as these assume independent clustering.

```
tabcom <- combineTests(clustered.sig$indices[[1]], result$table)
head(tabcom)

##   nWindows logFC.up logFC.down       PValue          FDR direction
## 1        1        0          0 8.232598e-04 9.948283e-04        up
## 2        1        0          0 1.338344e-03 1.351877e-03        up
## 3        1        0          0 8.615071e-04 1.020448e-03        up
## 4        1        0          0 2.545062e-04 4.645498e-04        up
## 5        1        0          0 7.417598e-05 2.046502e-04        up
## 6        1        1          0 3.303351e-06 2.440763e-05        up

tabbest <- getBestTest(clustered.sig$indices[[1]], result$table)
head(tabbest)

##   best     logFC   logCPM        F       PValue          FDR
## 1   36 0.2769275 6.177678 15.03136 8.232598e-04 9.948283e-04
## 2   42 0.3240413 3.959689 13.51290 1.338344e-03 1.351877e-03
## 3   43 0.3248683 4.333464 14.88638 8.615071e-04 1.020448e-03
## 4   44 0.3721554 4.340252 19.00829 2.545062e-04 4.645498e-04
## 5   50 0.4590971 3.342406 23.69257 7.417598e-05 2.046502e-04
## 6   51 0.5932889 3.582650 38.22647 3.303351e-06 2.440763e-05
```

Test statistics are saved to file for later examination, along with the coordinates of each cluster's bounding box. The combined p-value from `combineTests` is stored but is only used for sorting and should not be interpreted as a significance measure.

```
tabstats <- data.frame(tabcom[,1:4], logFC=tabbest$logFC, FDR=clustered.sig$FDR)
results.d <- as.data.frame(clustered.sig$interactions)[,useful.cols]
results.d <- cbind(results.d, tabstats)
o.d <- order(results.d$PValue)
write.table(results.d[o.d,], file="DIclusters.tsv", sep="\t",
            quote=FALSE, row.names=FALSE)
```

Again, the statistics can stored in the metadata of *GInteractions* object.

```
mcols(clustered.sig$interactions) <- cbind(
    mcols(clustered.sig$interactions), tabstats)
```

7.3 Merging results from different bin widths

7.3.1 Clustering with different bin widths

The optimal choice of bin size is not clear when there are both sharp and diffuse changes in the interaction space. Smaller bins provide greater spatial resolution and can identify sharp DIs that would be lost within larger bin pairs. Conversely, larger bin pairs have larger counts and greater power to detect diffuse DIs. Comprehensive detection of DIs can be achieved by combining analyses from several bin sizes. For example, `clusterPairs` accepts multiple *InteractionSet* objects to cluster results of analyses with different sizes.

```
clustered.mult <- clusterPairs(larger=data, smaller=smaller.data,
                               tol=1, upper=1e6)
head(clustered.mult$indices$larger)

## [1] 1 2 3 4 5 6

head(clustered.mult$indices$smaller)

## [1] 1 2 2 2 2 2
```

Alternatively, the `boxPairs` function can be used to identify all smaller bin pairs that are nested within each of the larger "parent" bin pairs. This is a form of independent clustering where all nested bin pairs are defined as a single cluster. Each set of nested bin pairs will only be reported once, reducing redundancy and limiting the potential for misinterpretation of the FDR. Note that the larger bin size *must* be an integer multiple of the smaller bin size(s). This is necessary to simplify the interpretation of the nesting procedure.

```
boxed <- boxPairs(larger=data, smaller=smaller.data)
head(boxed$indices$larger)

## [1] 1 2 3 4 5 6

head(boxed$indices$smaller)

## [1] 1 2 2 2 2 2
```

7.3.2 Computing consolidated cluster-level statistics

Regardless of whether `clusterPairs` or `boxPairs` is used, the statistics for each cluster can be computed with the `consolidatePairs` function. This uses a weighted version of Simes' method to ensure each analysis contributes equally to the significance of each cluster [32]. For each bin pair, the weight of its p-value is inversely proportional to the number of bin pairs of the same size in the same cluster. This ensures that the combined p-value calculation for each cluster is not dominated by smaller, more numerous bin pairs. The BH method is then applied to the combined p-values to control the cluster-level FDR.

```
merged.results <- list(result$table, result.small$table)
cons <- consolidatePairs(boxed$indices, merged.results)
head(cons)

##   nWindows logFC.up logFC.down       PValue          FDR direction
## 1        2        0          2 8.290943e-04 2.311825e-02      down
## 2       56         0          0 1.043286e-02 1.301009e-01        up
## 3      100         1         10 3.732473e-01 7.056014e-01     mixed
## 4       56         0          6 1.718373e-09 6.708121e-07      down
## 5      100        26          4 3.196865e-01 6.632161e-01        up
## 6      101         1         14 1.696180e-04 6.788497e-03      down

sum(cons$FDR <= 0.05)

## [1] 7619
```

Here, the number of detections is greater than that found with large bins alone. This suggests that the different bin sizes complement each other by detecting features at different resolutions. Statistics for each of the larger bin pairs can then be stored to file. Reordering is performed using the combined p-value to promote the strongest changes.

```
results.b <- data.frame(as.data.frame(boxed$interactions)[,useful.cols], cons)
o.b <- order(results.b$PValue)
write.table(results.b[o.b,], file="boxed.tsv", sep="\t",
            quote=FALSE, row.names=FALSE)
```

7.3.3 Clustering significant bin pairs of different widths

The `diClusters` function will also accept multiple *InteractionSet* objects in the form of a list. It will simply pass the arguments to `clusterPairs` and control the cluster-level FDR as described in Section 7.2.5. It will also use a frequency-weighted version of the FDR to ensure an equal contribution from each bin size when defining significant bin pairs.

```
merged.data <- list(data, smaller.data)
clustered.mult <- diClusters(merged.data, merged.results, target=0.05,
                             cluster.args=list(tol=1))
length(clustered.mult$interactions)

## [1] 11091
```

The output is also directly compatible with `consolidatePairs`. Again, users should ignore the p-values when clustering based on significant bin pairs.

```
cons.sig <- consolidatePairs(clustered.mult$indices, merged.results)
```

7.4 Reporting nested bin pairs

It is often convenient to identify the top-ranked bin pair nested within each of the larger features, i.e., parent bin pairs or clusters. Here, the top-ranked bin pair is identified by `getBestTest` as the one with the smallest individual p-value. This means that any high-resolution changes nested within a large feature can be easily identified. However, keep in mind that the FDR is computed with respect to the features, not the nested bin pairs.

```
inside <- getBestTest(boxed$indices$smaller, result.small$table)
best.interactions <- interactions(smaller.data)[inside$best,]
```

```
inter.frame <- as.data.frame(best.interactions)[,useful.cols[-c(1,4)]]
nested <- data.frame(inter.frame, inside[,c("logFC", "F")])
head(nested)
```

```
##    start1    end1  start2    end2      logFC         F
## 1 3004106 3100835       1 3004109 -0.7240461  4.571166
## 2 3004106 3100835 3004106 3100835  0.3347865 11.298605
## 3 4801990 4899085 3801896 3901860 -1.2225469  7.671230
## 4 4693997 4801993 4500265 4599273 -0.8968364 45.455462
## 5 5599281 5695146 3004106 3100835  1.0039228  9.068671
## 6 5001372 5100850 4801990 4899085 -0.5602859 23.252441
```

```
expanded <- rep(NA, nrow(results.b)) # For parents with no nested elements.
expanded[as.integer(rownames(inside))] <- seq_len(nrow(inside))
results.b <- data.frame(results.b, best=nested[expanded,])
write.table(results.b[o.b,], file="boxed_best.tsv", sep="\t",
            quote=FALSE, row.names=FALSE)
```

The above code only reports the top-ranked nested bin pair within each large feature. This may not be sufficient when many internal changes are occurring. An alternative approach is to store the entirety of the `smaller.data` in a *R* save file, along with `cons` and `data`. Any interesting nested changes can then be interactively identified for a given feature.

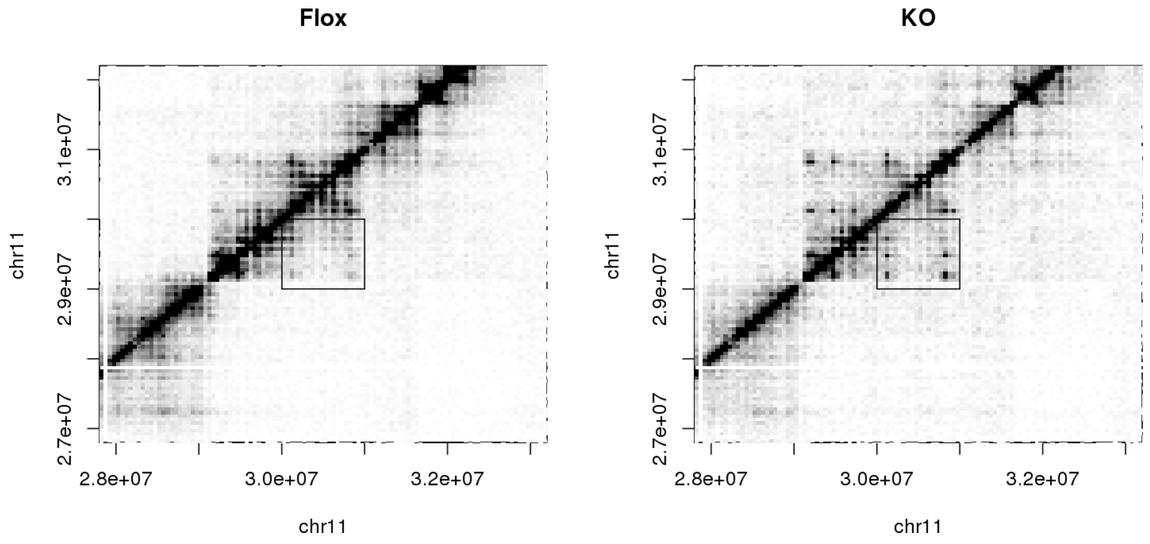
7.5 Visualization with plaid plots

7.5.1 Using conventional plaid plots

Plaid plots are widely used to visualize the distribution of read pairs in the interaction space [3]. In these plots, each axis is a chromosome segment. Each "pixel" represents an interaction between the corresponding intervals on each axis. The color of the pixel is proportional to the number of read pairs mapped between the interacting loci. We demonstrate below using large bin pairs detected in the consolidated analysis with small nested bin pairs.

```
# Setting up the interaction space to be plotted.
chosen <- o.b[1]
chosen.a1 <- anchors(boxed$interactions[chosen], type="first")
chosen.a2 <- anchors(boxed$interactions[chosen], type="second")
expanded1 <- resize(chosen.a1, fix="center", width=bin.size*5)
expanded2 <- resize(chosen.a2, fix="center", width=bin.size*5)
cap.wt <- 100
cap.ko <- cap.wt*data$totals[3]/data$totals[1]

# Plotting the WT library.
par(mfrow=c(1,2))
plotPlaid(input[1], first=expanded1, second=expanded2, max.count=cap.wt,
          width=5e4, param=mm.param, main="Flox")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))

# Plotting the KO library.
```

```
plotPlaid(input[3], first=expanded1, second=expanded2, max.count=cap.ko,
          width=5e4, param=mm.param, main="KO")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))
```



Expansion of the plot boundaries ensures that the context of the interaction can be determined by examining the features in the surrounding space. It is also possible to tune the size of the pixels through a parameter that is, rather unsurprisingly, named `width`. In this case, the side of each pixel represents a 50 kbp bin, rounded to the nearest restriction site. The actual bin pair occurs at the center of the plot and is marked by a rectangle.

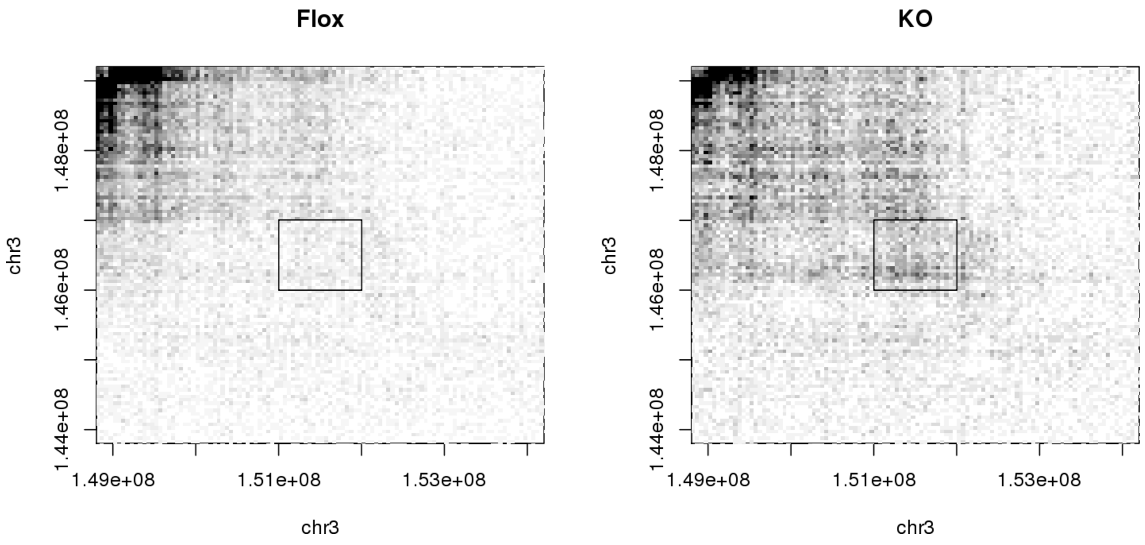
The `max.count` value controls the relative scale of the colors. Any pixel with a larger count will be set at the maximum color intensity. This ensures that a few high-count regions do not dominate the plot. A smaller `max.count` is necessary for smaller libraries so that the intensity of the colors is comparable. The actual color can be set by specifying `col`.

In the example above, the differential interaction is driven mainly by the smaller bin pairs. Changes in intensities are particularly prevalent at the top left and bottom right corners of the rectangle. By comparison, the fold change for the entire bin pair is a little less than 30%. This highlights the usefulness of including analyses with smaller bin sizes.

Another example is shown below for the top DI detected in the analysis using only large bins. Because the counts are "averaged" across the area of the interaction space, the change must be consistent throughout that area (and thus, more obvious) for detection to be successful. Of course, any sharp changes within each of these large bin pairs will be overlooked as the smaller bin pairs are not used.

```
# Setting up the interaction space to be plotted.
chosen <- o.r[1]
chosen.a1 <- anchors(data[chosen], type="first")
chosen.a2 <- anchors(data[chosen], type="second")
expanded1 <- resize(chosen.a1, fix="center", width=bin.size*5)
expanded2 <- resize(chosen.a2, fix="center", width=bin.size*5)
cap.wt <- 30
cap.ko <- cap.wt*data$totals[3]/data$totals[1]

# Plotting the WT sample.
```

```
par(mfrow=c(1,2))
plotPlaid(input[1], first=expanded1, second=expanded2, max.count=cap.wt,
          width=5e4, param=mm.param, main="Flox")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))

# Plotting the KO sample.
plotPlaid(input[3], first=expanded1, second=expanded2, max.count=cap.ko,
          width=5e4, param=mm.param, main="KO")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))
```

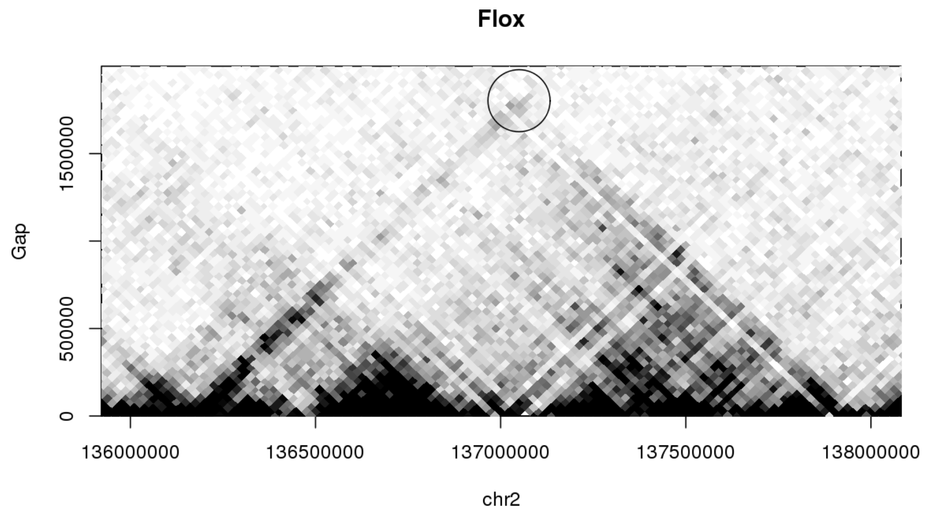


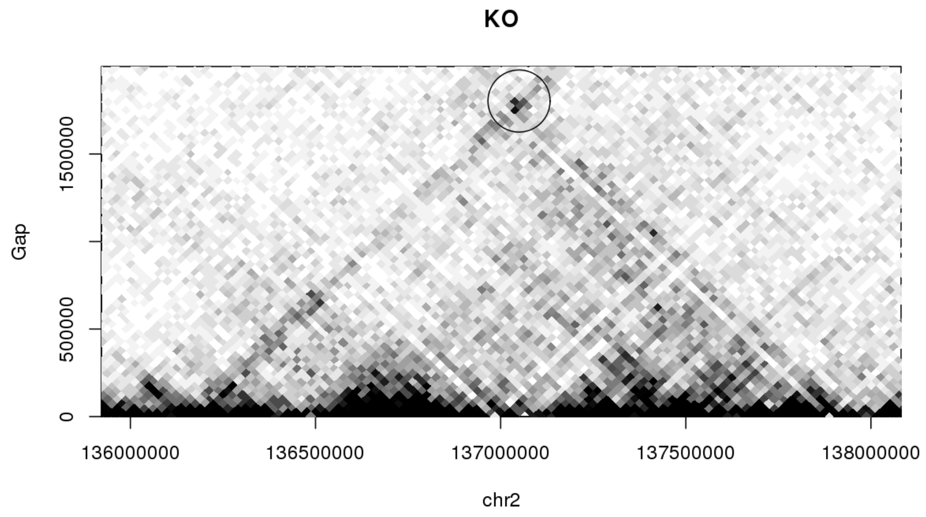
7.5.2 Using rotated plaid plots

Alternatively, users may prefer to use `rotPlaid` to generate rotated plaid plots. These are more space-efficient and are easier to stack onto other genomic tracks, e.g., for ChIP-seq data. However, rotated plots are only effective for local interactions within a specified region. Some more effort is also required in interpretation. In the example below, each colored box represents an interaction between two bins. The coordinates of each interacting bin can be identified by extending lines from opposite sides of the box until they intersect the x-axis.

```
# Setting up the interaction space to be plotted.
chosen <- o.b[4]
example <- anchors(boxed$interactions[chosen], type="second")
end(example) <- end(anchors(boxed$interactions[chosen], type="first"))
best.mid.a1 <- (results.b$best.start1[chosen]+results.b$best.end1[chosen])/2
best.mid.a2 <- (results.b$best.start2[chosen]+results.b$best.end2[chosen])/2
best.mid <- (best.mid.a1 + best.mid.a2)/2
best.gap <- best.mid.a1 - best.mid.a2

# Plotting the WT sample.
par(mfrow=c(2,1))
rotPlaid(input[1], mm.param, region=example, width=2.5e4,
         main="Flox", max.count=cap.wt)
```

```
points(best.mid, best.gap, cex=7)

# Plotting the KO sample.
rotPlaid(input[3], mm.param, region=example, width=2.5e4,
         main="KO", max.count=cap.ko)
points(best.mid, best.gap, cex=7)
```



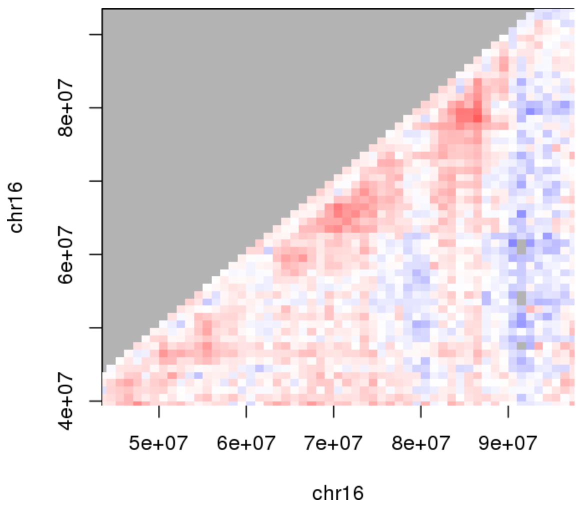


The circle marks the area of the interaction space that corresponds to the top-ranked nested bin pair within the `chosen` larger bin pair. An increase in the interaction intensity is clearly observed in the KO condition. This sharp change would not be observed with larger bin pairs, where the final count would be dominated by other (non-differential) areas.

7.5.3 Using differential plaid plots

In some cases, it may be more informative to display the magnitude of the changes across the interaction space. This is achieved using the `plotDI` function, which assigns colors to bin pairs according to the size and direction of the log-fold change. Visualization of the changes is useful as it highlights the DIs, whereas conventional plaid plots are dominated by high-abundance features like TADs. The latter features may be constant between libraries and, thus, not of any particular interest. The log-fold changes also incorporate normalization information, which is difficult to represent on a count-based plaid plot.

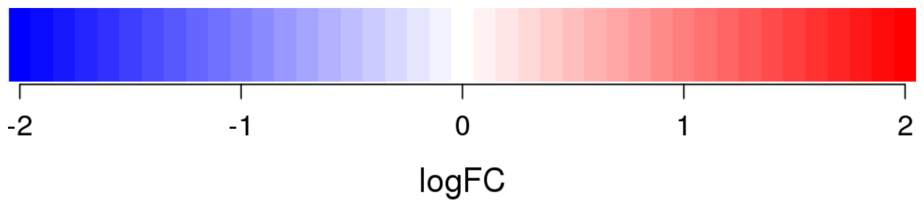
```
chosen <- o.r[5]
chosen.a1 <- anchors(data[chosen], type="first")
chosen.a2 <- anchors(data[chosen], type="second")
expanded1 <- resize(chosen.a1, fix="center", width=5e7)
expanded2 <- resize(chosen.a2, fix="center", width=5e7)
colfun <- plotDI(data, result$table$logFC, expanded1, expanded2, diag=FALSE)
```



The example above uses red and blue for positive and negative log-fold changes, respectively. White and near-white regions correspond to those with log-fold change close to zero. Grey regions mark the parts of the space where no bin pairs are present in `data`, possibly due to filtering on abundance. As a result, this approach tends to be less useful when high-abundance bin pairs are more sparsely distributed, e.g., for long-range interactions. A rotated DI plot can be similarly constructed using the `rotDI` function.

Both `rotDI` and `plotDI` will invisibly return another function that maps log-fold changes to colors. This can be used to generate a custom color bar, as shown below. A similar function that maps counts to colors is also returned by `plotPlaid` and `rotPlaid`.

```
logfc <- -20:20/10
plot(0,0,type="n", axes=FALSE, xlab="", ylab="", xlim=range(logfc), ylim=c(0,1))
rect(logfc - 0.05, 0, logfc + 0.05, 1, col=colfun(logfc), border=NA)
axis(1, cex.axis=1.2)
mtext("logFC", side=1, line=3, cex=1.4)
```


-2
-1
0
1
2
logFC

Chapter 8

Detecting differential domain boundaries

This chapter, like the cheese, stands alone, so there's not much we require from other chapters. We'll only need the `input` and `mm.param` objects from Chapter 3.

8.1 Overview

High intensity triangles are often observed on the diagonal of the intra-chromosomal interaction space. These correspond to topologically associating domains (TADs) where loci within each domain interact more frequently than those between domains. Such domains are proposed to control genomic behavior by limiting the scope for interactions and restraining the spread of chromatin marks [33]. At higher resolutions, small domains may also be formed due to looping between specific genomic elements [22].

To identify these domains, Dixon *et al.* defined a "directionality index" for each genomic locus [34]. This was computed by counting the number of read pairs mapped between the target locus and an "upstream" interval with higher genomic coordinates; counting the number of read pairs mapped between the target locus and a "downstream" interval with lower genomic coordinates; and taking the normalized difference of the counts. A region at the start of a domain will interact preferentially with upstream regions in the same domain, compared to downstream regions in a different domain. Conversely, the region at the end of each domain will interact preferentially with the downstream regions compared to upstream regions. This results in a characteristic pattern of positive directionality indices at the start of the domain, followed by negative values at the end. These patterns can be extracted with hidden Markov models (HMMs) to explicitly identify the domain boundaries.

An alternative analysis is to identify loci where the size or direction of the directionality statistic changes between conditions. This complements the standard DI analysis by focusing on regions where the domain boundary changes in strength or orientation. Thus, changes in domain definition between conditions can be quantified in a statistically rigorous manner.

8.2 Loading up- and downstream counts

Up- and downstream counts for each 100 kbp genomic bin are loaded using the `domainDirections` function. This returns a *RangedSummarizedExperiment* object, in which each row corresponds to a bin and each column corresponds to a library. The two matrices `"up"` and `"down"` contain the counts to the up- and downstream regions, respectively.

```
finder <- domainDirections(input, mm.param, width=1e5, span=10)
finder

## class: RangedSummarizedExperiment
## dim: 26729 4
## metadata(3): param span width
## assays(2): up down
## rownames: NULL
## rowData names(1): nfrags
## colnames: NULL
## colData names(0):
```

The two matrices are combined into a single matrix, and the total counts for each library are also loaded. These counts are used to construct a *DGEList* for analysis with *edgeR*.

```
all.counts <- cbind(assay(finder, "up"), assay(finder, "down"))
totals <- totalCounts(input, mm.param)
ydom <- DGEList(all.counts, lib.size=rep(totals, 2))
```

As an aside, the same counts can be used to compute the directionality index [34]. A Gaussian HMM can then be fitted to explicitly define domain boundaries, using packages like *depmixS4*. However, this analysis is largely outside the scope of *diffHic* and will not be discussed here.

8.3 Constructing the design matrix

A design matrix is constructed with condition-specific coefficients for the log-fold change between up- and downstream counts. It also contains sample-specific blocking factors to avoid incorporating unnecessary variability in the dispersion estimate due to sample-specific effects. Recall that each library contributes two sets of counts to the final matrix, so the vectors containing condition and sample information must be doubled appropriately.

```
Condition <- factor(c("flox", "flox", "ko", "ko"))
Sample <- factor(seq_along(input))
Condition <- rep(Condition, 2)
Sample <- rep(Sample, 2)
Direction <- rep(c("Up", "Down"), each=length(input))
design <- model.matrix(~0 + Sample + Direction:Condition)
```

Some further manipulation is performed to clean up the design matrix prior to downstream analysis. Redundant coefficients that cause linear dependencies are removed, and the remaining columns are given simpler names. For convenience, the condition-specific up/down log-fold changes will be referred to as directionality statistics in the text below.

```
design <- design[,!grepl("DirectionDown", colnames(design))]
colnames(design) <- sub("DirectionUp:", "", colnames(design))
design

##   Sample1 Sample2 Sample3 Sample4 Conditionflox Conditionko
## 1       1       0       0       0             1           0
## 2       0       1       0       0             1           0
## 3       0       0       1       0             0           1
## 4       0       0       0       1             0           1
## 5       1       0       0       0             0           0
## 6       0       1       0       0             0           0
## 7       0       0       1       0             0           0
## 8       0       0       0       1             0           0
```

8.4 Pre-processing of the count data

Filtering is performed to remove low-abundance bins that do not contain enough counts to reject the null hypothesis. In general, this should have little effect as most of the counts should be very large. This is because `domainDirections` effectively counts read pairs for highly local interactions (as the up- or downstream intervals are adjacent to each bin).

```
ab <- aveLogCPM(ydom)
keep <- ab > 0
ydom <- ydom[keep,]
summary(keep)

##    Mode   FALSE    TRUE
## logical    1602   25127
```

The same filter is applied to the set of genomic bins to match with the entries of `ydom`.

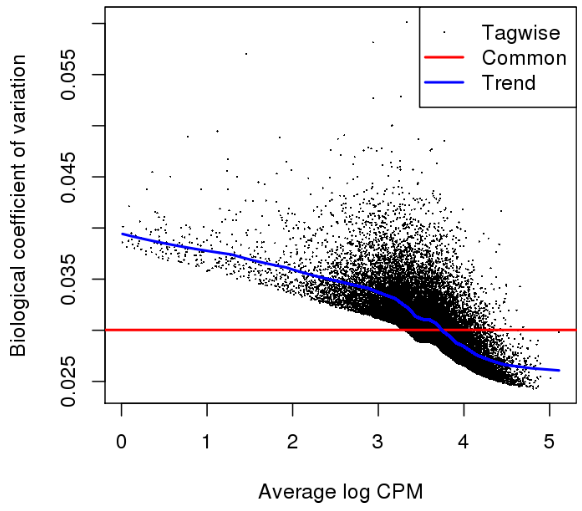
```
cur.regions <- rowRanges(finder)[keep,]
```

Normalization is not performed here as it is not required. The model contains sample-specific blocking factors, which largely negates the need for normalization between samples. Additionally, the differential test for each bin will compare directionality values between conditions. This means that any biases between the up- and downstream counts within each library or condition (e.g., due to mappability of different regions) should cancel out.

8.5 Testing for significant differences with *edgeR*

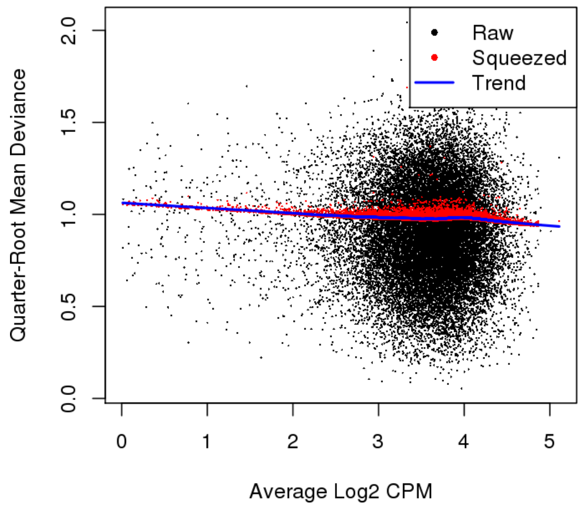
The analysis proceeds directly to dispersion estimation and GLM fitting. The mean-dependent trend in the NB dispersions is modelled using `estimateDisp` as previously described.

```
ydom <- estimateDisp(ydom, design)
plotBCV(ydom)
```



A GLM is fitted to the counts for each bin using the specified model and the trended NB dispersion, and QL dispersions are estimated for all bins using robust EB shrinkage.

```
fitdom <- glmQLFit(ydom, design, robust=TRUE)
plotQLDisp(fitdom)
```



The aim of the differential analysis is to identify bins where the directionality statistics change between conditions. The `makeContrasts` function is used to construct an appropriate contrast vector, and the QL F-test is applied to detect significant differences.

```
con <- makeContrasts(Conditionko - Conditionflox, levels=design)
resdom <- glmQLFTest(fitdom, contrast=con)
topTags(resdom)

## Coefficient:  -1*Conditionflox 1*Conditionko
##             logFC   logCPM         F       PValue          FDR
## 2386   0.9988340 4.162087 154.24808 8.060085e-17 2.025258e-12
## 4595  -1.1083619 3.422453 131.62341 1.511300e-15 1.898722e-11
## 1408   1.0645073 3.912424 132.28480 3.538199e-15 2.963478e-11
## 14712  0.8743536 4.239697 115.14725 1.628958e-14 1.023271e-10
## 5606   0.9730969 3.687062 111.38663 2.898853e-14 1.456789e-10
## 12073  0.8063455 4.229938 106.29182 6.470913e-14 2.709911e-10
## 1872  -0.7809661 4.451015 105.05435 7.896109e-14 2.834365e-10
## 16854 -0.8406668 4.108131 104.20904 9.054610e-14 2.843940e-10
## 757   -0.7928171 4.229549  96.66591 3.181203e-13 8.881567e-10
## 6323   0.8459912 3.929275  95.71750 3.742996e-13 9.100861e-10
```

The interpretation of these changes requires some care. If the directionality statistic was positive in the WT cells, a positive `logFC` would represent a strengthening of the domain boundary. However, if it was negative, a positive `logFC` would represent a weakening of the domain boundary or a reversal of the domain orientation. To assist interpretation, users are advised to report the condition-specific directionality statistics in the output.

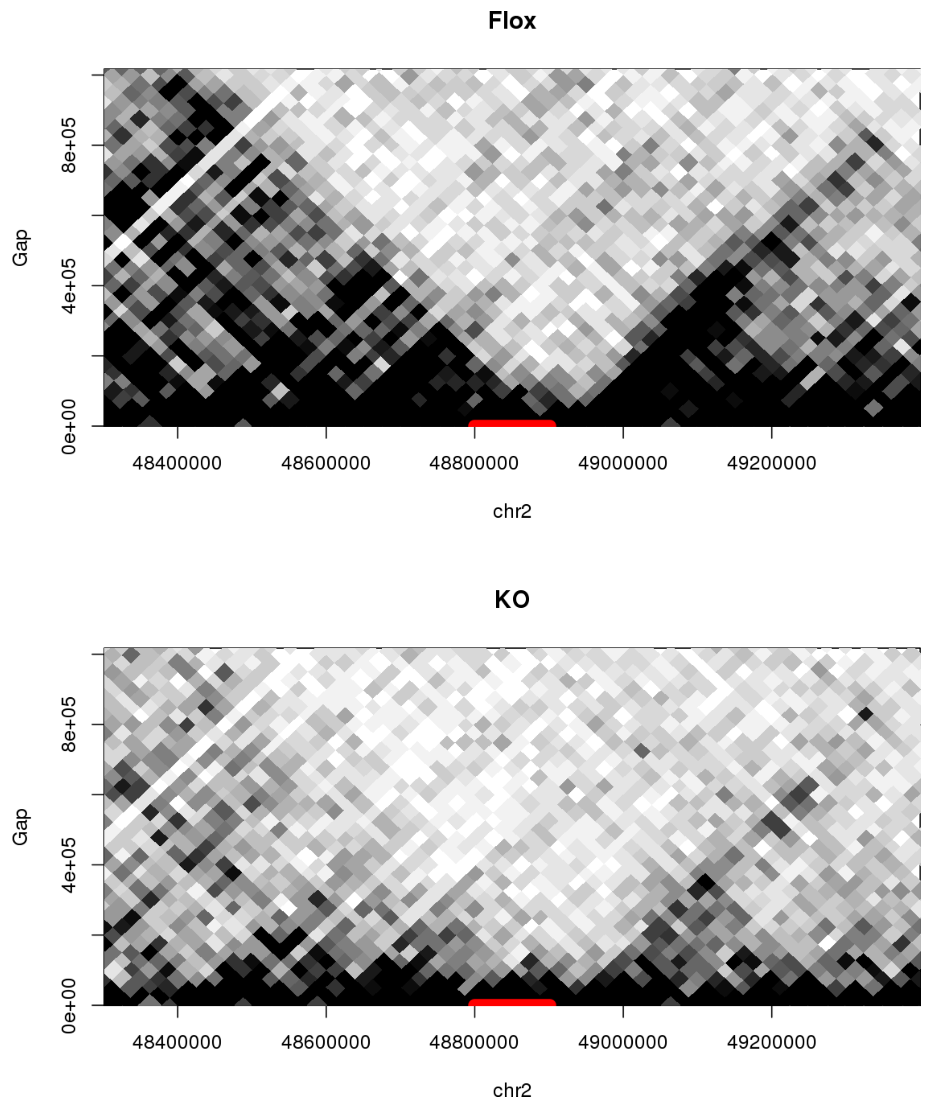
```
output <- data.frame(as.data.frame(cur.regions)[,1:3],
                     KO=fitdom$coefficients[,"Conditionko"]/log(2),
                     Flox=fitdom$coefficients[,"Conditionflox"]/log(2),
                     resdom$table)
output$FDR <- p.adjust(resdom$table$PValue, method="BH")
o <- order(output$PValue)
output <- output[o,]
head(output[,1:5], 10) # not showing test statistics for brevity

##       seqnames     start       end         KO       Flox
## 2386      chr2  48799546  48901284 -0.5048126 -1.5036466
## 4595      chr3  90499471  90598684 -0.8547325  0.2536294
## 1408      chr1 143599379 143696849 -0.2618755 -1.3263828
## 14712    chr10 105698492 105799539 -0.8688601 -1.7432137
## 5606      chr4  34499260  34602534 -0.2329656 -1.2060626
## 12073     chr8  89700733  89799215 -0.5422071 -1.3485526
## 1872      chr1 190000997 190100303  0.8932561  1.6742222
## 16854    chr12  73005689  73097744  0.7891691  1.6298359
## 757       chr1  78500961  78603837  0.5289130  1.3217301
## 6323      chr4 106298974 106398973  0.1908319 -0.6551592
```

Users can also examine whether the absolute values of the directionalities increase or decrease between conditions. This provides an indication of whether the domains become more or less well-defined. Here, domain definition seems to weaken in the KO condition.

```
is.sig <- output$FDR <= 0.05
summary(is.sig)

##    Mode   FALSE    TRUE
## logical   19339    5788

change.type <- abs(output$KO) > abs(output$Flox)
summary(change.type[is.sig])

##    Mode   FALSE    TRUE
## logical    4174    1614
```

Finally, the region with the top change in directionality is visualized. Each rotated plaid plot shows the surrounding interaction space, with the region itself highlighted in red.

```
tophit <- cur.regions[o[1]]
expanded <- resize(tophit, fix="center", width=width(tophit)*10)

par(mfrow=c(2,1))
rotPlaid(input[1], mm.param, region=expanded, width=2.5e4, main="Flox")
segments(start(tophit), 0, end(tophit), 0, col="red", lwd=10)
rotPlaid(input[3], mm.param, region=expanded, width=2.5e4, main="KO")
segments(start(tophit), 0, end(tophit), 0, col="red", lwd=10)
```



Chapter 9

Epilogue

Congratulations on getting to the end. As a reward for your efforts, here is a poem:

I once had a friend named Björk,
With him I would always talk,
But he was a pig,
So when he got big,
We killed him and ate his pork.

9.1 Data sources

All datasets are publicly available from the NCBI Gene Expression Omnibus (GEO). The K562/GM06990 dataset [3] was obtained using the GEO accession number GSE18199. The neural stem cell data set [4] was obtained using the accession GSE49017. Finally, the RWPE1 dataset [5] was obtained using the accession GSE37752. All libraries were processed as described in Chapter 2. For some datasets, multiple technical replicates are available for each library. These were merged together prior to read pair counting.

All libraries were downloaded in the Sequence Read Archive format (SRA). The SRA files were aligned to the reference genome using the '`sra2bam.sh`' script at https://github.com/LTLA/diffHicUsersGuide. Some software packages are also required to run this script, such as various tools from the *Picard* suite – read the source code in '`sra2bam.sh`' for more details. The resulting BAM files were then converted into index files using the '`bam2hdf.R`' script.

9.2 Session information

```
sessionInfo()
```

```
## R version 3.4.0 Patched (2017-04-28 r72639)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: CentOS release 6.4 (Final)
##
## Matrix products: default
## BLAS: /wehisan/home/allstaff/a/alun/Software/R/R-3-4-branch_devel/lib/libRblas.so
## LAPACK: /wehisan/home/allstaff/a/alun/Software/R/R-3-4-branch_devel/lib/libRlapack.so
##
## 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] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] csaw_1.11.3
##  [2] BiocParallel_1.11.9
##  [3] TxDb.Mmusculus.UCSC.mm10.knownGene_3.4.0
##  [4] GenomicFeatures_1.29.11
##  [5] AnnotationDbi_1.39.3
##  [6] BSgenome.Mmusculus.UCSC.mm10_1.4.0
##  [7] BSgenome.Hsapiens.UCSC.hg19_1.4.0
##  [8] BSgenome_1.45.3
##  [9] rtracklayer_1.37.3
## [10] Biostrings_2.45.4
## [11] XVector_0.17.1
## [12] edgeR_3.19.7
## [13] limma_3.33.13
## [14] diffHic_1.9.8
## [15] InteractionSet_1.5.7
## [16] SummarizedExperiment_1.7.10
## [17] DelayedArray_0.3.21
## [18] matrixStats_0.52.2
## [19] Biobase_2.37.2
## [20] GenomicRanges_1.29.15
## [21] GenomeInfoDb_1.13.5
## [22] IRanges_2.11.19
## [23] S4Vectors_0.15.12
## [24] BiocGenerics_0.23.3
##
## loaded via a namespace (and not attached):
##  [1] progress_1.1.2          statmod_1.4.30
##  [3] locfit_1.5-9.1          splines_3.4.0
##  [5] lattice_0.20-35         rhdf5_2.21.6
##  [7] htmltools_0.3.6         yaml_2.1.14
##  [9] blob_1.1.0              XML_3.98-1.9
## [11] rlang_0.1.2             DBI_0.7
## [13] bit64_0.9-7             GenomeInfoDbData_0.99.1
## [15] stringr_1.2.0           zlibbioc_1.23.0
## [17] evaluate_0.10.1         memoise_1.1.0
## [19] knitr_1.17              biomaRt_2.33.4
## [21] highr_0.6                 Rcpp_0.12.13
## [23] KernSmooth_2.23-15        backports_1.1.1
## [25] bit_1.1-12                Rsamtools_1.29.1
## [27] BiocStyle_2.5.40          digest_0.6.12
## [29] stringi_1.1.5             Rhtslib_1.9.2
## [31] grid_3.4.0                rprojroot_1.2
## [33] tools_3.4.0               bitops_1.0-6
## [35] magrittr_1.5              RCurl_1.95-4.8
## [37] RSQLite_2.0               tibble_1.3.4
## [39] pkgconfig_2.0.1           Matrix_1.2-11
## [41] prettyunits_1.0.2         assertthat_0.2.0
## [43] rmarkdown_1.6             R6_2.2.2
## [45] GenomicAlignments_1.13.6 compiler_3.4.0
```

Bibliography

[1] 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.

[2] A. T. L. Lun and G. K. Smyth. diffHic: a Bioconductor package package to detect differential genomic interactions in Hi-C data. *BMC Bioinformatics*, 16:258, 2015.

[3] E. Lieberman-Aiden, N. L. van Berkum, L. Williams, M. Imakaev, T. Ragoczy, A. Telling, I. Amit, B. R. Lajoie, P. J. Sabo, M. O. Dorschner, R. Sandstrom, B. Bernstein, M. A. Bender, M. Groudine, A. Gnirke, J. Stamatoyannopoulos, L. A. Mirny, E. S. Lander, and J. Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, Oct 2009.

[4] S. Sofueva, E. Yaffe, W. C. Chan, D. Georgopoulou, M. Vietri Rudan, H. Mira-Bontenbal, S. M. Pollard, G. P. Schroth, A. Tanay, and S. Hadjur. Cohesin-mediated interactions organize chromosomal domain architecture. *EMBO J.*, 32(24):3119–3129, Dec 2013.

[5] D. S. Rickman, T. D. Soong, B. Moss, J. M. Mosquera, J. Dlabal, S. Terry, T. Y. MacDonald, J. Tripodi, K. Bunting, V. Najfeld, F. Demichelis, A. M. Melnick, O. Elemento, and M. A. Rubin. Oncogene-mediated alterations in chromatin conformation. *Proc. Natl. Acad. Sci. U.S.A.*, 109(23):9083–9088, Jun 2012.

[6] M. Imakaev, G. Fudenberg, R. P. McCord, N. Naumova, A. Goloborodko, B. R. Lajoie, J. Dekker, and L. A. Mirny. Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nat. Methods*, 9(10):999–1003, Oct 2012.

[7] V. C. Seitan, A. J. Faure, Y. Zhan, R. P. McCord, B. R. Lajoie, E. Ing-Simmons, B. Lenhard, L. Giorgetti, E. Heard, A. G. Fisher, P. Flicek, J. Dekker, and M. Merkenschlager. Cohesin-based chromatin interactions enable regulated gene expression within preexisting architectural compartments. *Genome Res.*, 23(12):2066–2077, Dec 2013.

[8] M. Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, 17(1):10–12, 2011.

[9] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, 9(4):357–359, Apr 2012.

[10] J. M. Belton, R. P. McCord, J. H. Gibcus, N. Naumova, Y. Zhan, and J. Dekker. Hi-C: a comprehensive technique to capture the conformation of genomes. *Methods*, 58(3):268–276, Nov 2012.

[11] F. Jin, Y. Li, J. R. Dixon, S. Selvaraj, Z. Ye, A. Y. Lee, C. A. Yen, A. D. Schmitt, C. A. Espinoza, and B. Ren. A high-resolution map of the three-dimensional chromatin interactome in human cells. *Nature*, 503(7475):290–294, Nov 2013.

[12] J. C. Marioni, C. E. Mason, S. M. Mane, M. Stephens, and Y. Gilad. RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome Res.*, 18(9):1509–1517, Sep 2008.

[13] W. Ma, F. Ay, C. Lee, G. Gulsoy, X. Deng, S. Cook, J. Hesson, C. Cavanaugh, C. B. Ware, A. Krumm, J. Shendure, C. A. Blau, C. M. Disteche, W. S. Noble, and Z. Duan. Fine-scale chromatin interaction maps reveal the cis-regulatory landscape of human lincRNA genes. *Nat. Methods*, 12(1):71–78, Jan 2015.

[14] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, Mar 2010.

[15] A. T. Lun, M. Perry, and E. Ing-Simmons. Infrastructure for genomic interactions: Bioconductor classes for Hi-C, ChIA-PET and related experiments. *F1000Res.*, 5:950, 2016.

[16] J. R. Hughes, N. Roberts, S. McGowan, D. Hay, E. Giannoulatou, M. Lynch, M. De Gobbi, S. Taylor, R. Gibbons, and D. R. Higgs. Analysis of hundreds of cis-regulatory landscapes at high resolution in a single, high-throughput experiment. *Nat. Genet.*, 46(2):205–212, Feb 2014.

[17] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, Sep 2012.

[18] 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.

[19] 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.

[20] W. A. Bickmore. The spatial organization of the human genome. *Annu. Rev. Genomics Hum. Genet.*, 14:67–84, 2013.

[21] Y. C. Lin, C. Benner, R. Mansson, S. Heinz, K. Miyazaki, M. Miyazaki, V. Chandra, C. Bossen, C. K. Glass, and C. Murre. Global changes in the nuclear positioning of genes and intra- and interdomain genomic interactions that orchestrate B cell fate. *Nat. Immunol.*, 13(12):1196–1204, Dec 2012.

[22] S. S. Rao, M. H. Huntley, N. C. Durand, E. K. Stamenova, I. D. Bochkov, J. T. Robinson, A. L. Sanborn, I. Machol, A. D. Omer, E. S. Lander, and E. L. Aiden. A 3D Map of the Human Genome at Kilobase Resolution Reveals Principles of Chromatin Looping. *Cell*, 159(7):1665–1680, Dec 2014.

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

[24] A. T. Lun and G. K. Smyth. csaw: a Bioconductor package for differential binding analysis of ChIP-seq data using sliding windows. *Nucleic Acids Res.*, 44(5):e45, Mar 2016.

[25] C. Loader. *Local Regression and Likelihood*. Statistics and Computing. Springer New York, 1999. URL: http://books.google.com.au/books?id=D7GgBAfL4ngC.

[26] 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.

[27] B. Phipson, S. Lee, I. J. Majewski, W. S. Alexander, and G. K. Smyth. Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *Ann. Appl. Stat.*, 10(2):946–963, 2016.

[28] Y. Chen, A. T. L. Lun, and G. K. Smyth. Differential expression analysis of complex RNA-seq experiments using edgeR. In S. Datta and D. S. Nettleton, editors, *Statistical Analysis of Next Generation Sequence Data*. Springer, New York, 2014.

[29] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. Roy. Statist. Soc. B*, pages 289–300, 1995.

[30] 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.*, May 2014.

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

[32] Y. Benjamini and Y. Hochberg. Multiple hypotheses testing with weights. *Scand. J. Stat.*, 24:407–418, 1997.

[33] E. P. Nora, J. Dekker, and E. Heard. Segmental folding of chromosomes: a basis for structural and regulatory chromosomal neighborhoods? *Bioessays*, 35(9):818–828, Sep 2013.

[34] J. R. Dixon, S. Selvaraj, F. Yue, A. Kim, Y. Li, Y. Shen, M. Hu, J. S. Liu, and B. Ren. Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature*, 485(7398):376–380, May 2012.