diffHic: Differential analysis of Hi-C data

User's Guide

Aaron Lun

First edition 12 December 2012

Last revised 7 October 2015

Contents

Chapter 1

Introduction

1.1 Scope

This document describes the analysis of Hi-C data with the diffHic package. Differential interactions are defined as those with significant changes in intensity between conditions. These are identified in a statistically rigorous manner using the methods in the edgeR package [Robinson et al., 2010]. 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 [Lun and Smyth, 2015].

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 was originally developed by Lieberman-Aiden et al. [2009]. It 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 performed whereby ligation between blunt ends in the same complex is favored. The ligated DNA is sonicated, and the sheared fragments containing ligation junctions are purified by a streptavidin pulldown. These purified ligation products are then subjected to paired-end sequencing. Mapping of the reads in each pair can identify the pairs of interacting loci. 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 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

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

3. filtering out uninteresting bin pairs

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

4. normalizing counts between libraries

```
> y <- asDGEList(data)
> y$offset <- normOffsets(data, type="loess")
```

5. modelling biological variability

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

6. testing for significant differences between groups

```
> result <- glmQLFTest(fit)
```

In the various examples for this guide, data will be used from three studies. The first dataset is the smallest and examines the chromatin structure in K562 and GM06990 cell lines [Lieberman-Aiden et al., 2009]. The second compares interaction intensities between wild-type and cohesin-deficient murine neural stem cells [Sofueva et al., 2013]. The final study compares ERG-overexpressing RWPE1 cells with a GFP-expressing control [Rickman et al., 2012]. 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 [Imakaev et al., 2012] or read splitting [Seitan et al., 2013]. The latter defines the ligation signature as the sequence that is obtained after ligation between blunt ends derived from cleaved restriction sites. For example, the *Hin*dIII 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 [Martin, 2011], 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 [Langmead and Salzberg, 2012]. Mapping by read splitting can be performed in diffHic using a custom Python script, below.

```
> 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).

```
> require(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 larger index is designated as the "anchor" whereas the smaller is the "target". This is demonstrated using Hi-C data from GM06990 cells.

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

$pairs
   total   marked filtered   mapped
 7068675   103594  1657451  5337797

$same.id
   dangling self.circle
     423205      135004

$singles
[1] 0

$chimeras
  total  mapped   multi invalid
2297481 1607255 1039918   67813
```

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 Picard's MarkDuplicates. Filtering on the minimum mapping quality score with `minq` is also recommended to remove spurious alignments.

Read pairs mapping to the same restriction fragment provide little information on interactions between fragments [Belton et al., 2012]. 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.

2.3 Processing of chimeric reads

For `preparePairs`, chimeric reads are handled by recording 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) will be used to determine the restriction fragment index for that read pair. The total number of chimeric read pairs will be reported, along with the number where $5'$ ends

or 3′ ends are mapped. 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.

The proportion of invalid chimeric pairs can then be calculated. 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] 0.06520995
```

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

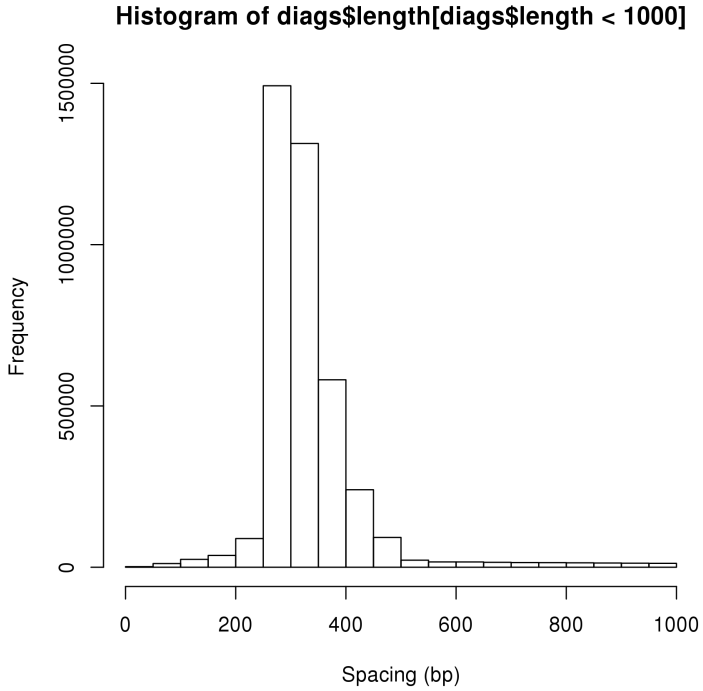
The `prunePairs` function removes read pairs that correspond to artifacts in the Hi-C procedure. The returned vector contains the number of read pairs removed for each artifact. Values of `length`, `inward` and `outward` correspond to removal by `max.frag`, `min.inward` and `min.outward`, respectively. 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
 4779588   858188    93665    82129  3758098
```

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 inferred by summing the distance between the mapping location of the 5′ end of each read and the nearest restriction site on the 3′ end of that read. Excessively large lengths are indicative of offsite 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)")
```



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 [Jin et al., 2013]. Such read pairs are technical artifacts that are (incorrectly) retained by `preparePairs`, as the two reads involved are mapped to different restriction fragments.

2.4.2 Setting size thresholds with strand orientation plots

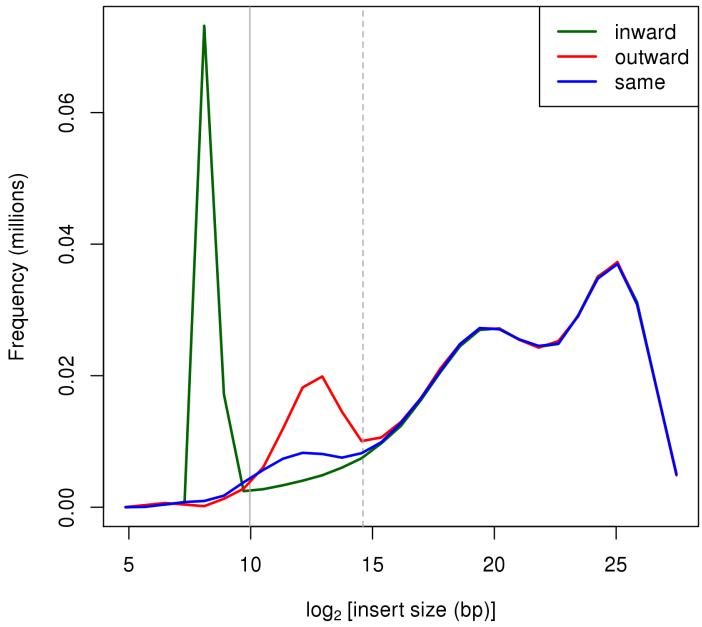
The strand orientation for a read pair refers to the combination of strands for the anchor/target reads. These are stored as flags where setting `0x1` or `0x2` means that the anchor or target reads, respectively, are mapped on the reverse strand. If different pieces of DNA were randomly ligated together, 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
750009 746583 746615 746502
```

This can be repeated for intra-chromosomal read pairs, by plotting the distribution of insert sizes for each strand orientation [Jin et al., 2013]. 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.

```
> 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$counts/2
> #
> ymax <- max(inward$counts, outward$counts, samestr$counts)/1e6
> xmax <- max(inward$mids, outward$mids, samestr$mids)
> xmin <- min(inward$mids, outward$mids, samestr$mids)
> #
> 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 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 length of the fragments after sonication, i.e., those actually used in sequencing. 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 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 [Marioni et al., 2008]. An example is provided below that merges several technical replicates for a GM06990 library in the Lieberman-Aiden et al. dataset.

```
> 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 anchor and target indices, as well as any additional information that might be useful. The idea is to allow entry 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`.

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

For full compatibility, users should include the alignment positions and lengths as `xxx.pos` and `xxx.len` for both the anchor and target reads (replacing `xxx` with `anchor` or `target`). 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 technique, whereby the DNase I enzyme is used to fragment the genome instead of restriction enzymes [Ma et al., 2015]. This allows for resolution beyond that of individual restriction fragments. However, cleavage sites for DNase I cannot be easily predicted to construct `param$fragments`. Formation of index files is subsequently inappropriate, as there are no restriction fragments for reads to be assigned to.

To handle this type of data in diffHic, a workaround is provided using the concept of "pseudo-fragments". The genome is partitioned into contiguous and non-overlapping pseudo-fragments of constant size, using the `segmentGenome` function. Reads can then be assigned into each pseudo-fragment in the same manner as that to actual restriction fragments for standard Hi-C. This is done using the `prepPseudoPairs` function, which is (almost) equivalent to running `preparePairs` with the pseudo-fragment coordinates in `param$fragments`.

An example of this approach is shown below. For demonstration purposes, the SRR027957 library will be treated as DNase Hi-C sample. The hg19 genome is segmented into 500 bp pseudo-fragments, and reads are then assigned to each of these pseudo-fragments.

```
> seg.frags <- segmentGenome(BSgenome.Hsapiens.UCSC.hg19, size=500)
> prepPseudoPairs("SRR027957.bam", pairParam(seg.frags), file="SRR027957.h5",
+     dedup=TRUE, minq=10)

$pairs
   total   marked filtered   mapped
 7068675   103594  1657451  5337797
```

```
$singles
[1] 0

$chimeras
  total  mapped   multi invalid
2297481 1656727 1072925   41444
```

Unlike `preparePairs`, no diagnostic information regarding self-circles or dangling ends is reported. Such metrics are based on restriction fragment coordinates, but these are not experimentally relevant for artificial pseudo-fragments. Similarly, the length of the sequencing fragment is computed from artificial fragment boundaries and has little meaning. This means that the `length` field should be ignored in the output of `getPairData`. 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.

This pseudo-fragment strategy allows easy passage through the diffHic pipeline. While some spatial resolution is lost, this is largely irrelevant in downstream steps. For example, counting across the interaction space will use regions much larger than the pseudo-fragments. Any loss of resolution from pseudo-fragment assignment is likely to be negligble.

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 [Li and Durbin, 2010]. A Python script for iterative mapping [Imakaev et al., 2012] 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 Sofueva et al. dataset. 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.

```
> require(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 bin pairs can be 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

DIList object for 4 libraries with 3319100 pairs across 2739 regions

> head(counts(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

> head(anchors(data))

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(targets(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 [      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
```

This generates a `DIList` object that contains the relevant count and coordinate information. Each row of the count matrix represents the counts for an interaction, while each column represents a library. Each interaction is characterized as that between a pair of genomic intervals - in this case, genomic bins. Again, anchor and target notation is used for these intervals, whereby the anchor bin is that with the "higher" genomic coordinate.

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 7.2.5 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.

```
> require(TxDb.Mmusculus.UCSC.mm10.knownGene)
> gene.body <- genes(TxDb.Mmusculus.UCSC.mm10.knownGene)
```

Counting can be 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 as 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
```

```
DIList object for 4 libraries with 12672322 pairs across 23653 regions
```

Again, anchor and target notation applies whereby the interval with the larger start coordinate in the genome is defined as the 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 3 metadata columns:
         seqnames              ranges strand |     gene_id    nfrags  original
            <Rle>           <IRanges>  <Rle> | <character> <integer> <integer>
  497097     chr1 [3211067, 3675240]      * |      497097       197     15169
   19888     chr1 [4286646, 4409818]      * |       19888        49      6944
   20671     chr1 [4487488, 4498343]      * |       20671         2      7332
   27395     chr1 [4772601, 4786029]      * |       27395         3     12945
   18777     chr1 [4802092, 4847111]      * |       18777         8      6374
   21399     chr1 [4856126, 4899085]      * |       21399        16      8019
  -------
  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 [Hughes et al., 2014]. 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))

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(targets(viewdata))

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         1
```

```
[2]     chr1 [3004106, 3100835]      * |        34         1         2
[3]     chr1 [3100832, 3194461]      * |        30         1         3
[4]     chr1 [3194458, 3301641]      * |        41         1         4
[5]     chr1 [3301638, 3399158]      * |        52         1         5
[6]     chr1 [3399155, 3501374]      * |        39         1         6
-------
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.

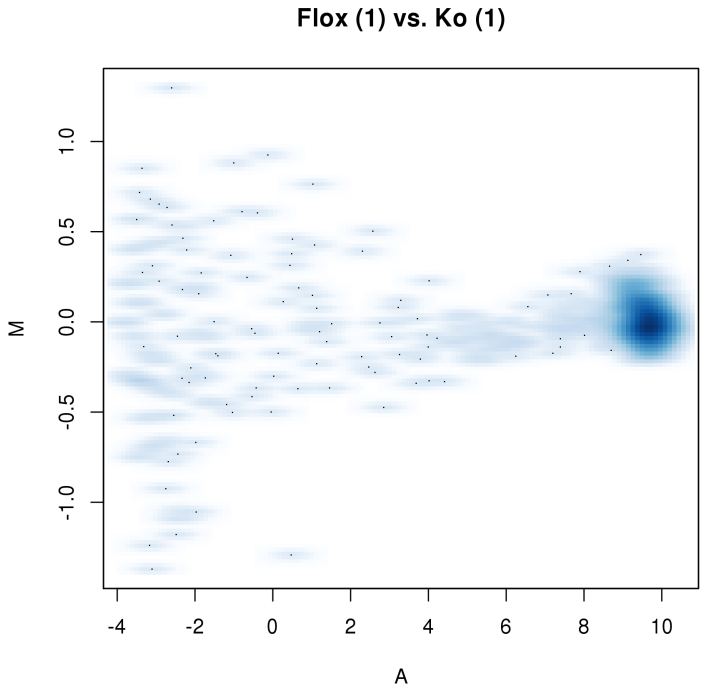
```
> margin.data <- marginCounts(input, mm.param, width=bin.size)
> margin.data

DIList object for 4 libraries with 2732 pairs across 2739 regions
```

Each row of the returned `DIList` contains the marginal read counts for a single genomic bin. Note that the anchor/target notation is superfluous for `marginCounts`. This is because the marginal counts refer to individual bins rather than bin pairs. Nonetheless, a `DIList` is still returned for consistency, in which the anchor and target regions are identical.

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 reliable. 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 elect to restrict counting to particular chromosomes, by setting a value for 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.

```
> 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 choose to discard alignments that lie within blacklisted regions, using 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 [Consortium, 2012], and can be obtained by following this **link**. 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., 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 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`.

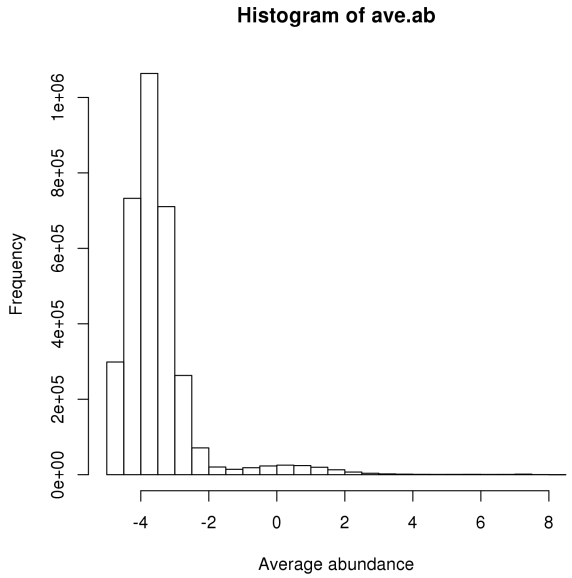
4.1 Overview

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 [Bourgon et al., 2010]. 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 [McCarthy et al., 2012].

```
> require(edgeR)
> ave.ab <- aveLogCPM(asDGEList(data))
> hist(ave.ab, xlab="Average abundance")
```



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 interfere with the asymptotic approximations that are required for downstream statistical modelling. The example below identifies those bin pairs with an overall NB mean not less than 5.

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

   Mode   FALSE    TRUE    NA's
logical 2433389  885711       0
```

The `count.keep` vector is then used to subset the `DIList`. 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

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 can be 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 [Bickmore, 2013]. The median abundance across inter-chromosomal bin pairs is used as the estimate of the non-specific ligation rate. Here, the threshold is defined as a minimum fold change of 5 above this estimate.

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

[1] -3.845944

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

   Mode   FALSE    TRUE    NA's
logical 3159187  159913       0
```

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
```

4.3 Filtering as a function of interaction distance

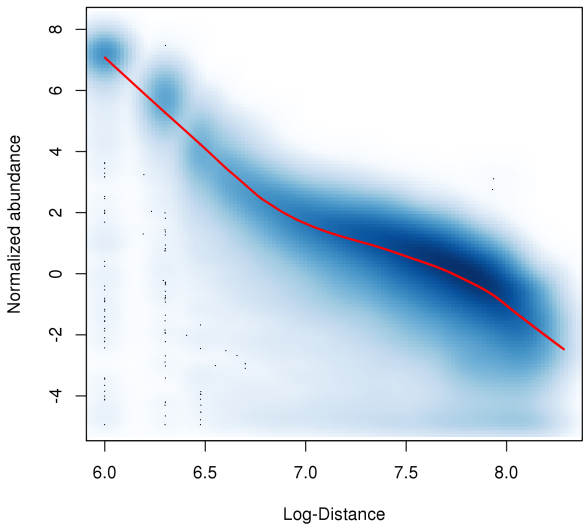
A more complex filter adjusts the threshold according to the distance between the bins in each bin pair. In typical Hi-C datasets, larger counts are observed at lower interaction distances. This is probably driven by non-specific compaction of chromatin into a 3D "globule" conformation. If such compaction is uninteresting, a concomitantly higher threshold is necessary to offset the increase in counts for these local interactions [Lin et al., 2012].

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 [Lieberman-Aiden et al., 2009]. 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    NA's
logical 1445284 1873816       0
```

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    NA's
logical 3081407  237693       0
```

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
```

Note that the distance between bins can also be obtained directly with the `getDistance` 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.

4.4 Peak-calling in the interaction space

4.4.1 Defining enrichment values for each bin pair

Peaks in the interaction space refer to those bin pairs that have substantially more reads than their neighbors. The `enrichedPairs` function computes an enrichment value for each bin pair, defined as the log-fold change of its abundance against that of its neighbors. Those bin pairs with high enrichment values can then be retained. This extends the concept of peak calling on the linear genome to the 2-dimensional interaction space [Rao et al., 2014].

```
> flank.width <- 5
> enrichments <- enrichedPairs(data, flank=flank.width)
> summary(enrichments)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-11.58000  -0.46120  -0.16740  -0.22830   0.08722  10.32000
```

More specifically, the function partitions the neighborhood into several different regions around the target bin pair. The size of the neighborhood is set by `flank`, and is defined in terms of bins (actual distance of $\sim$5 Mbp, in this case). The average abundance across each region is computed, and the region with the largest abundance is chosen. The enrichment value is defined as the log-fold change between the abundances of the target bin pair and the chosen region. Peaks are then defined as those bin pairs with high enrichment values. The neighborhood regions are designed to capture high-intensity structural features like looping domains, TADs or banding patterns [Rao et al., 2014]. Any bin pair inside a feature will be compared to other high-abundance bin pairs from the same feature. This avoids spuriously high enrichments due to comparisons with bin pairs outside that feature.

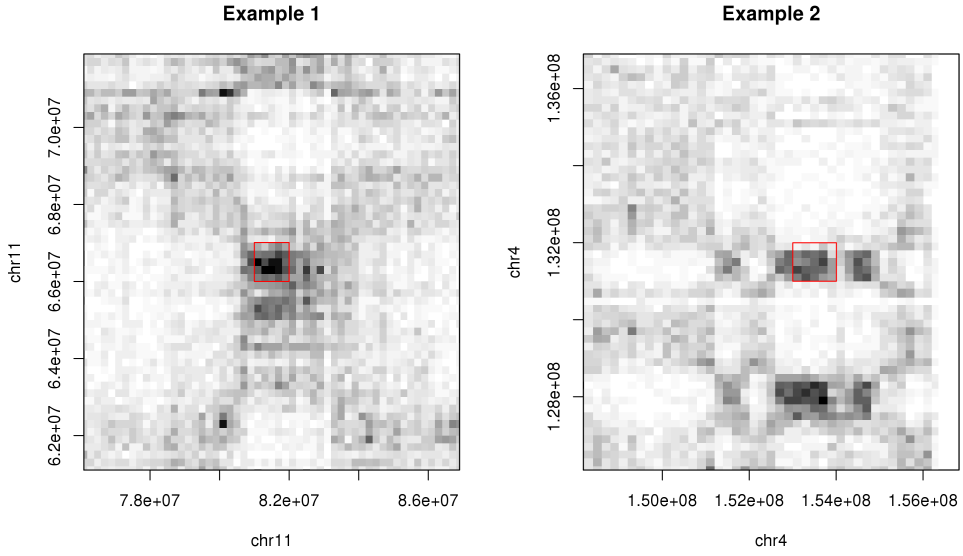
In this example, an enrichment value above 0.5 is required for retention. This is a modest threshold that errs on the side of caution, as weak peaks may be DIs and should be retained for testing. In practice, filtering of enrichment values should be combined with filtering on absolute abundances. This 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] 90049
```

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 7.3). 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 (i.e., less than 5) of nested bin pairs. This is because each peak, by definition, should be isolated in its neighborhood. Larger proportions suggest that more aggressive filtering on the enrichment score is required, e.g., in the second example.

```
> neighborhood <- (2*flank.width + 1) * exptData(data)$width
> boxed <- boxPairs(data[peak.keep], reference=neighborhood)
> out <- tabulate(tabulate(boxed$indices[[1]]))
> setNames(round(c(out[1:5], sum(out[5:length(out)]))/sum(out)*100, 1), c(1:5, ">5"))

   1    2    3    4    5   >5
21.7 22.0 18.2 13.3  9.1 24.9

> 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]]))
> setNames(round(c(out[1:5], sum(out[5:length(out)]))/sum(out)*100, 1), c(1:5, ">5"))

   1    2    3    4    5   >5
 5.4  4.3  4.0  3.7  3.7 82.7
```

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 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 enrichment statistic for that bin pair. 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 their enrichment values. This reduces memory usage at the cost of some speed.

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

DIList object for 4 libraries with 1060845 pairs across 26729 regions

> head(en.data$enrichment)

[1] -4.1027453  0.6462892 -0.7700094 -0.2075542 -1.0503617 -0.9616102
```

4.5 Defining filters in special cases

4.5.1 Computing filter thresholds with limited memory

These filtering procedures assume that no 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 lead to overestimation of the filter thresholds. 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. In such cases, several options are available:

- Pick an arbitrary threshold and use it to directly filter on the average abundances. This is simple but the chosen threshold has no obvious interpretation.
- Load counts for each pair of chromosomes separately, using the `restrict` argument as described in Section 3.5.1. Abundances (and distances) can be computed for each pair and collected, to define the filter threshold without loading all counts into memory. Counts for each chromosome pair can be reloaded for separate filtering, and the remaining bin pairs can be aggregated for downstream analysis with `c(...)`. This is quite slow. Also, when computing abundances, the full library size should be manually specified in `asDGEList`. This allows valid comparisons between abundances from different chromosome pairs. Otherwise, the library size will be different for each pair.

```
> chr1.data <- squareCounts(input, width=1e5, filter=1,
+     param=reform(mm.param, restrict=cbind("chr1", "chr1")))
> totals <- totalCounts(input, mm.param)
> chr1.ab <- aveLogCPM(asDGEList(chr1.data, lib.size=totals))
```

- Load counts for larger bin sizes without pre-filtering. This reduces memory usage as the interaction space is partitioned into fewer bin pairs. Then, perform filtering and convert the computed filter thresholds into values for the original (smaller) bin sizes.

An example of the third option is shown here. For this section, imagine that a bin size of 100 kbp is actually of interest, but filter thresholds can only be efficiently computed with 1 Mbp bins. The first task is to load the counts for the smaller bin pairs. A non-unity value of `filter` is set in `squareCounts` to avoid using too much memory.

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

Direct filtering of these smaller bin pairs can be performed using `filterDirect`, with assistance from their larger counterparts. Counts for the 1 Mbp bin pairs are supplied as the `reference` parameter and are used to estimate the filter threshold. This threshold is then adjusted for differences in size between the smaller and larger bins. The final value can then be used to directly filter on the abundances of the smaller bin pairs. Here, a minimum 5-fold change over the threshold is required for retention of each 100 Mbp bin pair.

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

[1] -5.088111

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

   Mode   FALSE    TRUE    NA's
logical  425794  635051       0
```

The same approach can be used for trended filtering of small bin pairs. However, some extra effort is required here, as the trend is fitted to the abundances and distances of the larger bin pairs. Interpolation (and, if necessary, extrapolation) of the trend is performed to the distances of the smaller bin pairs. The interpolated values can then be used to filter on the abundances of the smaller bin pairs. This entire procedure 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    NA's
logical  616092  444753       0
```

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.

4.5.2 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  -4.755  -4.786  13.310
```

4.5.3 Filtering out diagonal elements

Incomplete removal of artifacts (e.g., dangling ends, self-circles) will generally manifest as increased 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 rest of the points. Such irregularities cannot be smoothly normalized and should be removed prior to further analysis, or analyzed separately. Removal can be achieved by generating the `ndiag.keep` logical vector for filtering, as shown below.

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

   Mode   FALSE    TRUE    NA's
logical    2614 3316486       0
```

Obviously, this will also remove short-range interactions that might be of interest. Users are advised to use a smaller bin size to recover these interactions. Short-range interactions will now be represented by the off-diagonal bin pairs, due to the improved spatial resolution of smaller bins. Artifacts will still be restricted to diagonal bin pairs, so long as the bins are larger than $\sim$25 kbp (see Section 2.4.2). While count sizes will also decrease, this should not be a major problem as counts should be larger at low interaction distances.

4.6 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 the irrelevance of non-specific ligation is indisputable. 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 like TMM normalization [Robinson and Oshlack, 2010]. This accounts for composition biases by assuming that most interactions do not change between conditions. Normalization factors are computed to scale the library sizes, yielding effective sizes that can be used in the downstream differential analysis.

```
> normfacs <- normOffsets(data)
> normfacs
```

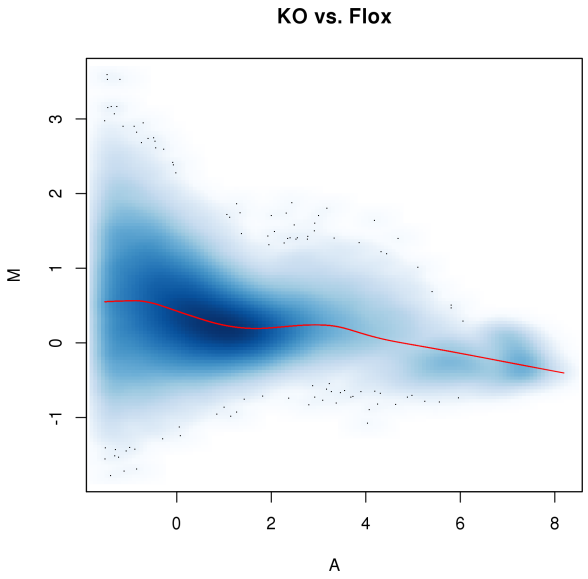
```
[1] 0.8953459 0.8979718 1.1127467 1.1177640
```

In practice, this scaling approach is 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

Trended biases can be generated from uncontrolled differences in library preparation. This is particularly problematic for Hi-C data, given the complexity of the 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 replicates.

```
> 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 vs. Flox")
> 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. The example below is based on a function from the csaw package, adapted from existing non-linear methods to handle discrete count data. An offset term is computed for each bin pair in each library, for use in a generalized linear model (GLM). A large offset for a count is equivalent to downscaling that count relative to the counts of other libraries.

```
> nb.off <- normOffsets(data, type="loess")
> head(nb.off)
```

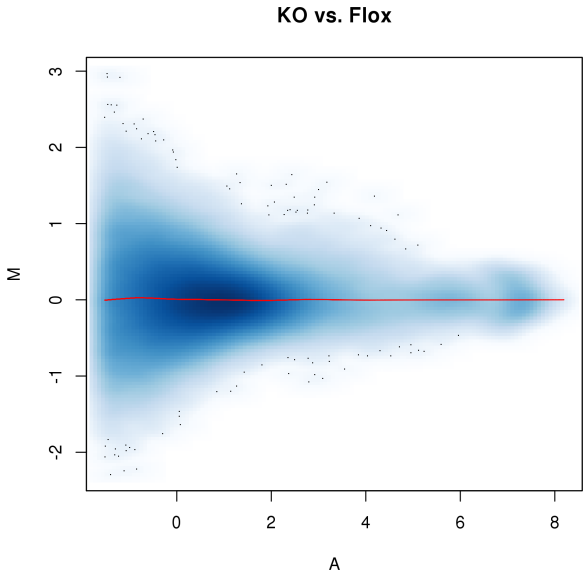
```
          [,1]       [,2]       [,3]        [,4]
[1,] 0.1359035 -0.1111409  0.0520197 -0.07678226
[2,] 0.3827167  0.2058267 -0.2489948 -0.33954867
[3,] 0.3021307  0.1516470 -0.1829528 -0.27082491
[4,] 0.3536789  0.1864192 -0.2252649 -0.31483322
[5,] 0.2285073  0.1019900 -0.1225945 -0.20790279
[6,] 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")
> nb.off[!neardiag] <- normOffsets(data[!neardiag,], type="loess")
```

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

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



Filtering on average abundance prior to this normalization step is strongly recommended. This removes low counts and avoids problems with discreteness. 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.

Of course, the non-linear strategy assumes that most 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, the simpler scaling strategy described in Section 5.1 may be more appropriate.

5.3 Iterative correction of interaction intensities

While this is not the intended function of the diffHic package, a method is also provided for the removal of biases between genomic regions. The `correctedContact` function performs the iterative correction procedure described by Imakaev et al. [2012] with some modifications. Briefly, if multiple libraries are used to generate the `data` object, then correction is performed using the overall NB mean for each bin pair. 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] 0.45657431 0.43319606 0.14754771 0.15232460 0.26722481 0.03447859
```

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] 186.415143   8.018365   2.848421   1.703012   1.324768   1.170795
 [7]   1.098713   1.060888   1.039164   1.025852   1.017334   1.011734
[13]   1.007989   1.005459   1.003739   1.002564   1.001760   1.001209
[19]   1.000831   1.000571   1.000392   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)
> anchor.bias <- corrected$bias[anchors(original, id=TRUE),]
> target.bias <- corrected$bias[targets(original, id=TRUE),]
> iter.off <- log(anchor.bias * target.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. This is done using the `normalizeCNV` function, based on multi-dimensional smoothing across several covariates with the `locfit` package [Loader, 1999]. It requires both bin pair and marginal counts, obtained with the same values of `width` and `param` in `squareCounts` and `marginCounts`, respectively.

```
> 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. Note that the marginal counts should be collected with the same parameters as the interaction counts. 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 Rickman et al. dataset 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,]
```

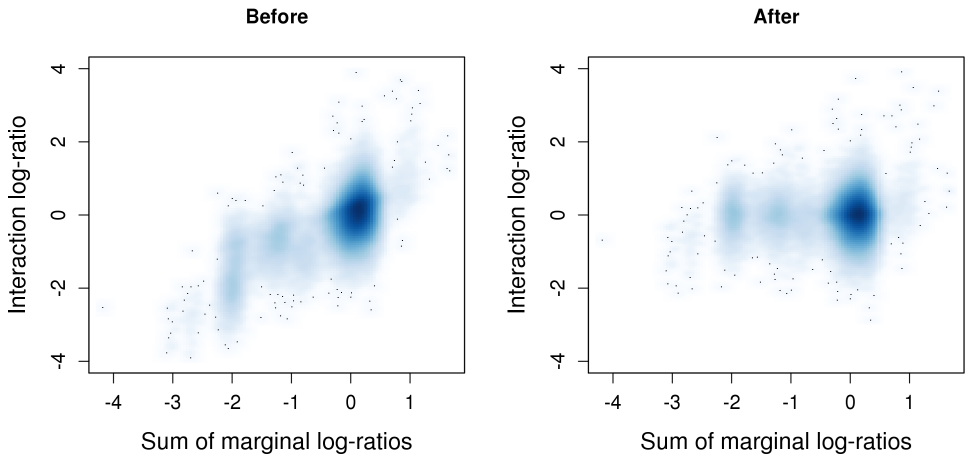
The aim is to plot the log-ratio of the interaction counts of each bin pair, as a function of the marginal log-ratios of the corresponding bins. This will illustrate the effect of the CNV on the interaction intensities between the two libraries. As two marginal log-ratios are present for each bin pair, these are added together for simplicity.

```
> matched <- matchMargins(rick.data, rick.marg)
> m.adjc <- cpm(asDGEList(rick.marg), log=TRUE)
> sum.madjc <- m.adjc[matched$amatch,] + m.adjc[matched$tmatch,]
> margin.lr <- sum.madjc[,1] - sum.madjc[,2]
```

Plotting can be performed to determine the effect of `normalizeCNV`. The presence of a trend indicates that decreases in copy number result in decreases in interaction intensity. After normalization, the trend in the interaction log-ratios is removed. Note that increasing `maxk` may be necessary to obtain sufficient accuracy in the internal call to `locfit`.

```
> before <- cpm(asDGEList(rick.data), log=TRUE)
> after <- log2(counts(rick.data)+0.5) - normalizeCNV(rick.data, rick.marg, maxk=1000)/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")
```



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 [Robinson et al., 2010]. 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. Specifically, variability is modelled in edgeR by estimating the NB dispersion parameter. 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 account for heteroskedasticity [Lund et al., 2012].

Dispersion estimation requires the fitting of a GLM to the counts for each bin pair [McCarthy et al., 2012]. To do so, a design matrix must be specified to describe the experimental setup. For the Sofueva et al. 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.

It is also necessary to assemble a `DGEList` object for entry into edgeR. Note the inclusion of the normalization offsets that were previously computed with the NB-loess method.

```
> y <- asDGEList(data)
> y$offset <- nb.off
```

6.2 Estimating the NB dispersion

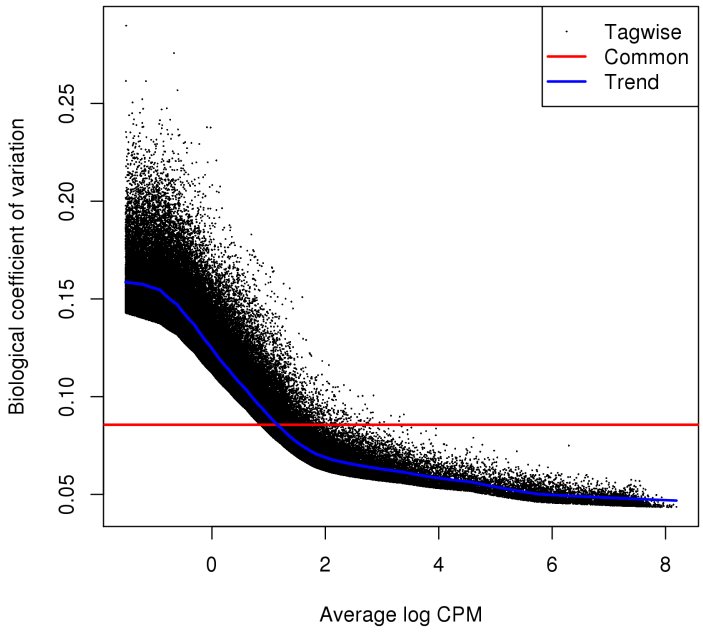
Estimation of the NB dispersion is performed by maximizing the Cox-Reid adjusted profile likelihood (APL) [McCarthy et al., 2012] for each bin pair. Of course, when replication is limited, there is not enough information per bin pair to estimate the dispersion. This is overcome by computing and sharing 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.

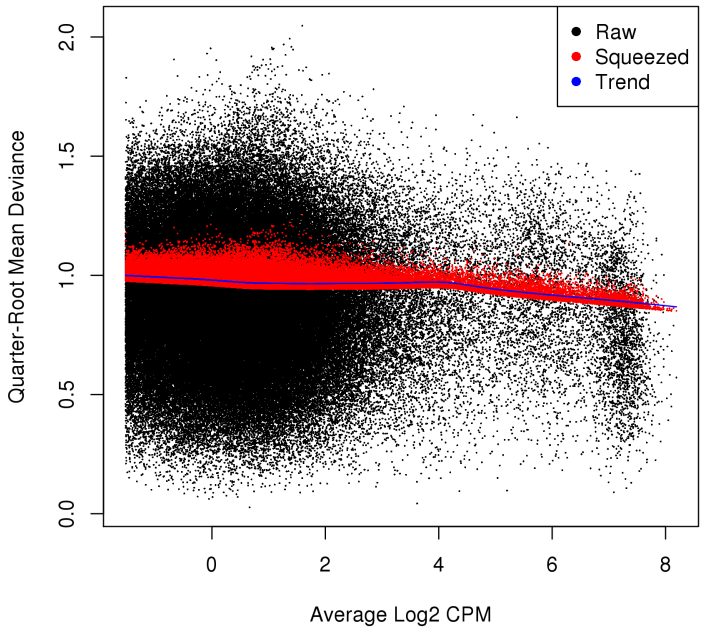
6.3 Estimating the QL dispersion

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, the QL dispersion can account for heteroskedasticity between bin pairs, whereas the NB dispersion cannot. Estimation of the former is 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 heteroskedasticity in the data. 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 if the true dispersions are not variable. This variability is quantified as the prior degrees of freedom, for which smaller values correspond to more heteroskedasticity and less shrinkage.

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.81   19.83   19.83   19.83   19.83   19.83
```

It is important to use the `robust=TRUE` argument in `glmQLFit`. This protects against any 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. In both cases, such outliers would inflate the apparant heteroskedasticity and increase 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 [Chen et al., 2014].

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 check that the contrast has been specified correctly through 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 1.383845 4.179442 231.3377 4.236141e-13 3.898619e-08
84768 1.540143 3.178067 228.1092 4.875924e-13 3.898619e-08
84766 1.540037 2.434434 166.5360 1.084390e-11 5.344619e-07
2541  1.385413 2.998188 160.3312 1.566248e-11 5.344619e-07
58458 1.031684 4.679999 159.2592 1.671102e-11 5.344619e-07
84767 1.203459 3.329989 150.3639 2.905290e-11 7.743229e-07
93501 1.075536 3.856085 146.0326 3.843289e-11 8.779884e-07
84765 1.418007 2.319288 141.4698 5.202048e-11 1.028270e-06
59485 1.054100 4.014325 138.9983 6.150945e-11 1.028270e-06
2542  1.047169 4.103016 138.3497 6.430185e-11 1.028270e-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 [Lund et al., 2012]. This means that it can maintain type I error control in the presence of heteroskedasticity, whereas the LRT does not.

7.2 Multiplicity correction and the FDR

7.2.1 Overview

Many bin pairs are tested for differences across the interaction space. Correction for multiple testing is necessary to avoid detection of many spurious differences. For genome-wide analyses, this correction can be performed by controlling the false discovery rate (FDR) with the Benjamini-Hochberg (BH) method [Benjamini and Hochberg, 1995]. This provides a suitable comprimise between specificity and sensitivity. In contrast, traditional methods of correction (e.g., Bonferroni) are often too conservative.

7.2.2 Direct application of the BH method

The BH method can be applied directly to the p-values for the individual bin pairs. In this case, the FDR refers to the proportion of detected bin pairs that are false positives. Significantly specific interactions are defined as those that are detected at an FDR of 5%.

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

```
[1] 4251
```

These can be saved to file as necessary. Resorting by p-value just makes it easier to parse the final table, as the most interesting differential interactions are placed at the top.

```
> ax.d <- anchors(data)
> tx.d <- targets(data)
> results.d <- data.frame(anchor.chr=seqnames(ax.d), anchor.start=start(ax.d),
+     anchor.end=end(ax.d),target.chr=seqnames(tx.d), target.start=start(tx.d),
+     target.end=end(tx.d), result$table, FDR=adj.p)
> o.d <- order(result$table$PValue)
> write.table(results.d[o.d,], file="results.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

7.2.3 Independent clustering of adjacent bin pairs

Adjacent bin pairs can be aggregated into larger clusters to reduce redundancy in the interpretation of the results. This is especially useful at smaller bin sizes, where multiple bin pairs may overlap a single underlying interaction. To demonstrate, a quick-and-dirty differential analysis of the previously loaded counts for the 100 kbp bin pairs is performed here.

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

The `clusterPairs` function puts two bin pairs in the same cluster if they are no more than `tol` bp apart. Cluster-level p-values are obtained by combining p-values across all bin pairs in the cluster using Simes' method [Simes, 1986]. Clusters can then be reported instead of individual bin pairs. Misinterpretation of the FDR is also mitigated as each interaction is represented by one cluster with one combined p-value (see Section 7.2.5).

```
> clustered <- clusterPairs(smaller.data, tol=1, upper=1e6)
> tabcluster <- csaw::combineTests(clustered$indices[[1]], result.small$table)
> head(tabcluster)
```

```
  nWindows logFC.up logFC.down       PValue          FDR
1        1        0          1 3.274575e-02 1.404891e-01
2       55        0          0 3.409829e-02 1.451059e-01
3       99        1         10 2.888988e-01 5.425381e-01
4       55        0          6 2.145027e-09 1.964833e-07
5       99       26          4 1.578324e-01 3.882597e-01
6      100        1         14 2.520249e-04 2.751694e-03
```

In practice, this approach requires aggressive filtering to avoid chaining effects. Otherwise, clustering will be confounded by the density of the interaction space, particularly at short distances and/or in TADs. Some protection is provided by setting `upper` to limit the maximum dimensions of each cluster. Even so, the boundaries of each cluster become ambiguous and difficult to interpret, e.g., if a whole TAD is absorbed into a cluster.

There is 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 7.2.6).

```
> ax.c <- clustered$anchors
> tx.c <- clustered$targets
> results.c <- data.frame(anchor.chr=seqnames(ax.c), anchor.start=start(ax.c),
+     anchor.end=end(ax.c), target.chr=seqnames(tx.c), target.start=start(tx.c),
+     target.end=end(tx.c), tabcluster)
> o.c <- order(tabcluster$PValue)
> write.table(results.c[o.c,], file="clustered.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

7.2.4 Clustering based on significant bin pairs

The interaction space is often dominated by high-abundance structural features like domains and compartments. Attempting to cluster on abundance may result in excessive chaining, yielding very large clusters for which the coordinates are mostly uninformative with respect to DIs. To avoid this, it may be necessary to cluster on 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.

To demonstrate, the test results for the 1 Mbp bin pairs will be used here. Significant bin pairs are defined at a FDR threshold of 5%, and clustered together with `clusterPairs`. Each cluster represents the coordinates of a DI. This is more useful than reporting the coordinates for an entire domain if only a portion of the domain is changing. No limits are placed on the maximum dimensions, as the sparsity of bin pairs should avoid chaining effects.

```
> threshold <- 0.05
> is.sig <- adj.p <= threshold
> sig.data <- data[is.sig,]
> clustered.sig <- clusterPairs(sig.data, tol=1, upper=NULL)
> head(clustered.sig$anchors)

GRanges object with 6 ranges and 0 metadata columns:
      seqnames                ranges strand
         <Rle>             <IRanges>  <Rle>
  [1]     chr1 [ 10000179,  13999307]      *
  [2]     chr1 [ 65001912,  71999866]      *
  [3]     chr1 [ 71003076,  74999308]      *
  [4]     chr1 [107998729, 112002471]      *
  [5]     chr1 [141998437, 151001683]      *
  [6]     chr1 [145998240, 148000612]      *
  -------
  seqinfo: 66 sequences from an unspecified genome

> head(clustered.sig$targets)

GRanges object with 6 ranges and 0 metadata columns:
      seqnames                ranges strand
         <Rle>             <IRanges>  <Rle>
  [1]     chr1 [  5997482,  11998773]      *
  [2]     chr1 [ 59996262,  68997516]      *
  [3]     chr1 [ 66989248,  71999866]      *
  [4]     chr1 [106000326, 112002471]      *
  [5]     chr1 [139996439, 145000108]      *
  [6]     chr1 [121997037, 124999713]      *
  -------
  seqinfo: 66 sequences from an unspecified genome
```

However, this summarization procedure is not statistically rigorous. Controlling the FDR across bin pairs does not guarantee control of the FDR over clusters [Lun and Smyth, 2014].

Instead, an informal estimate of the cluster-level FDR can be obtained with the `clusterFDR` function. The idea is to adjust `threshold` such that the cluster-level FDR lies below some desired value. All clusters formed at the chosen `threshold` can then be reported.

```
> csaw::clusterFDR(clustered.sig$indices[[1]], threshold)
```

```
[1] 0.1359846
```

In general, direct application of the BH method or independent clustering should be preferred to this *ad hoc* method. Users should only choose this method if direct application yields too many redundant results, or if the independent clusters are too large. Some loss of statistical rigour may be a necessary price to pay for interpretable results.

7.2.5 Merging results from different bin widths

The 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 differential interactions that would be lost within larger bin pairs. Conversely, larger bin pairs provide larger counts and greater power to detect diffuse events. Comprehensive detection of differential interactions can be achieved by combining analyses from several bin sizes.

To combine the results, the `boxPairs` function is used to identify all smaller bin pairs that are nested within each of the larger "parent" bin pairs. Specifically, smaller bin pairs are nested in a large bin pair if the former has the same ID as the latter in their respective `cons$id` vectors. 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
```

Each parent bin pair is associated with its own p-value and those of the smaller nested bin pairs. All of these p-values can be combined into a single p-value for the parent. To do this, the `consolidatePairs` function uses a weighted version of Simes' method [Benjamini and Hochberg, 1997]. For each bin pair, the weight of the p-value is inversely proportional to the number of bin pairs of the same size nested in the same parent. This ensures that the combined p-value calculation is not dominated by smaller, more numerous bin pairs. The BH method is then applied to the combined p-values to control the FDR across all parents.

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

```
  nWindows logFC.up logFC.down       PValue          FDR
1        2        0          2 8.879706e-04 2.481617e-02
2       56        0          0 1.860006e-02 1.839412e-01
3      100        1         10 4.962879e-01 8.038695e-01
4       56        0          6 4.290055e-09 1.698702e-06
5      100       26          4 3.156648e-01 6.811437e-01
6      101        1         14 5.040498e-04 1.629924e-02
```

```
> sum(cons$FDR <= 0.05)
```

```
[1] 7481
```

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.

```
> ax.s <- boxed$anchors
> tx.s <- boxed$targets
> o.s <- order(cons$PValue)
> results.s <- data.frame(anchor.chr=seqnames(ax.s), anchor.start=start(ax.s),
+     anchor.end=end(ax.s), target.chr=seqnames(tx.s), target.start=start(tx.s),
+     target.end=end(tx.s), cons)
> write.table(results.s[o.s,], file="sizecons.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

Nesting also reduces redundancy in the results. The set of nested bin pairs will only be reported once, in the form of the parent bin pair. This limits the potential for misinterpretation of the FDR [Lun and Smyth, 2014]. Similarly, for sparse data, users can also use the `clusterPairs` function to replace `boxPairs` in the above call. This will compute a combined p-value for each cluster, based on the p-values of the bin pairs of varying sizes in each cluster. The cluster-level statistics can then be reported for easier interpretation.

7.2.6 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 as `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.

```
> small.ids <- boxed$indices$smaller
> inside <- csaw::getBestTest(small.ids, result.small$table)
> ax.n <- anchors(smaller.data[inside$best,])
> tx.n <- targets(smaller.data[inside$best,])
> nested <- data.frame(anchor.start=start(ax.n), anchor.end=end(ax.n),
+     target.start=start(tx.n), target.end=end(tx.n),
+     result.small$table[inside$best,c("logFC", "F")])
> head(nested)
```

```
    anchor.start anchor.end target.start target.end      logFC         F
1        3004106    3100835            1    3004109 -0.7240191  4.559033
4        3100832    3194461      3100832    3194461  0.3960267 10.916939
180      4801990    4899085      3801896    3901860 -1.2225347  7.655951
170      4693997    4801993      4500265    4599273 -0.8968597 45.273688
351      5599281    5695146      3004106    3100835  1.0039041  9.081786
228      5001372    5100850      4801990    4899085 -0.5602828 23.204762
```

```
> expanded <- rep(NA, nrow(results.s)) # As some parents do not have nested elements.
> expanded[as.integer(rownames(inside))] <- 1:nrow(inside)
> results.n <- data.frame(results.s, nest=nested[expanded,])
> write.table(results.n[o.s,], file="withnest.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.

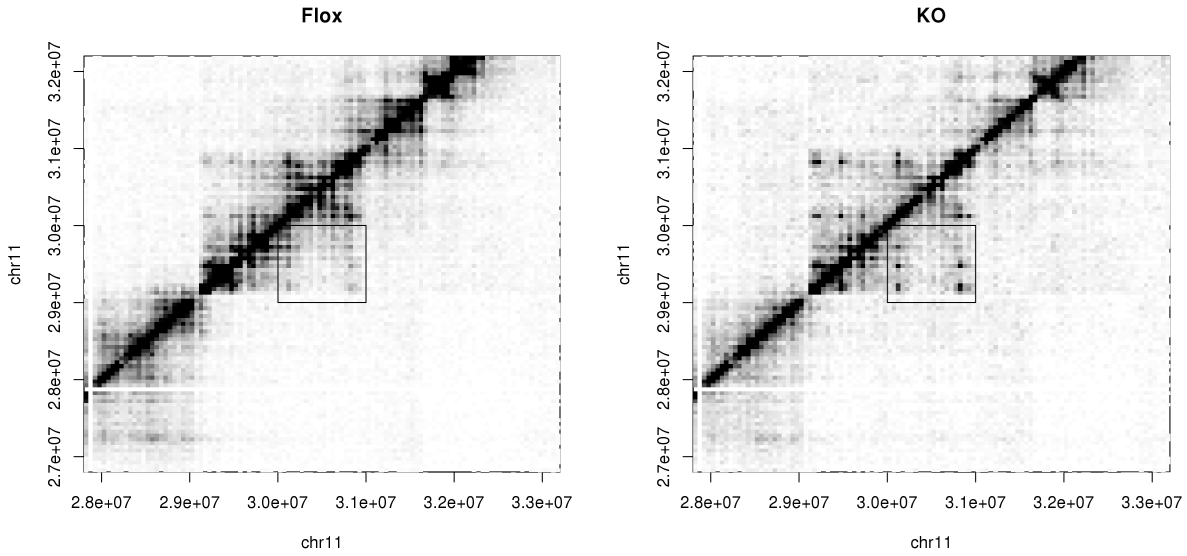
7.3 Visualization with plaid plots

7.3.1 Using conventional plaid plots

Plaid plots can be used to visualize the distribution of read pairs in the interaction space [Lieberman-Aiden et al., 2009]. Briefly, 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.

```
> chosen <- o.s[1]
> expanded.a <- resize(ax.s[chosen], fix="center", width=bin.size*5)
> expanded.t <- resize(tx.s[chosen], fix="center", width=bin.size*5)
> bound1 <- 100
> bound3 <- bound1*data$totals[3]/data$totals[1]
> plotPlaid(input[1], first=expanded.a, second=expanded.t, max.count=bound1,
+     width=5e4, param=mm.param, main="Flox")
> rect(start(ax.s[chosen]), start(tx.s[chosen]), end(ax.s[chosen]), end(tx.s[chosen]))
> plotPlaid(input[3], first=expanded.a, second=expanded.t, max.count=bound3,
+     width=5e4, param=mm.param, main="KO")
> rect(start(ax.s[chosen]), start(tx.s[chosen]), end(ax.s[chosen]), end(tx.s[chosen]))
```

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.

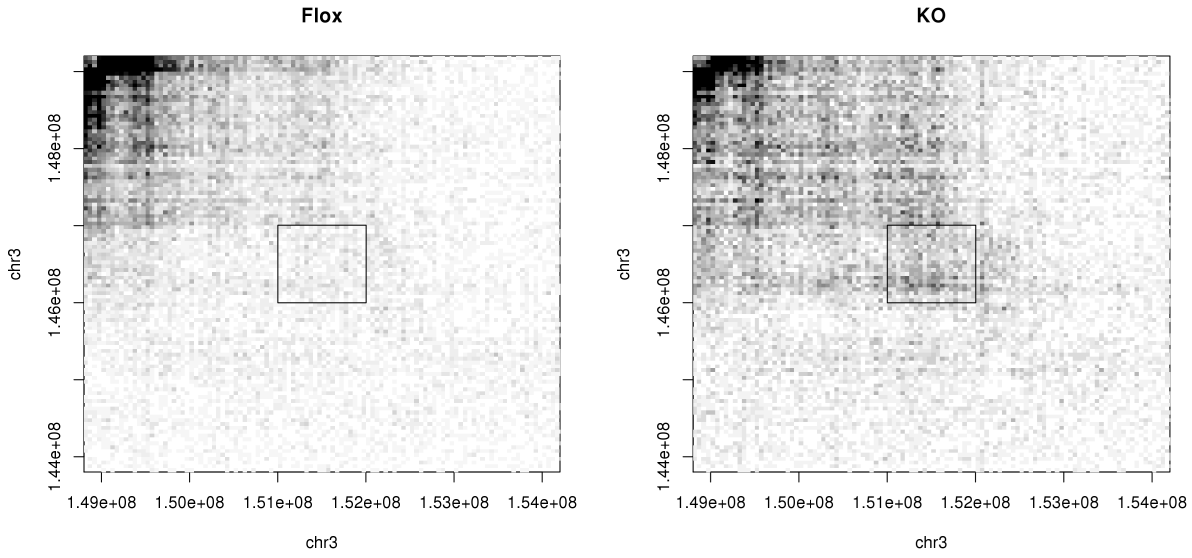


The `max.count` value controls the relative scale of the colors. Any pixel with a larger count will be set at the maximum color intensity. This ensures that a few high-count regions do not dominate the plot. A smaller `max.count` is necessary for smaller libraries so that the intensity of the colors 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. The following plots are constructed for the top differential interaction 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.

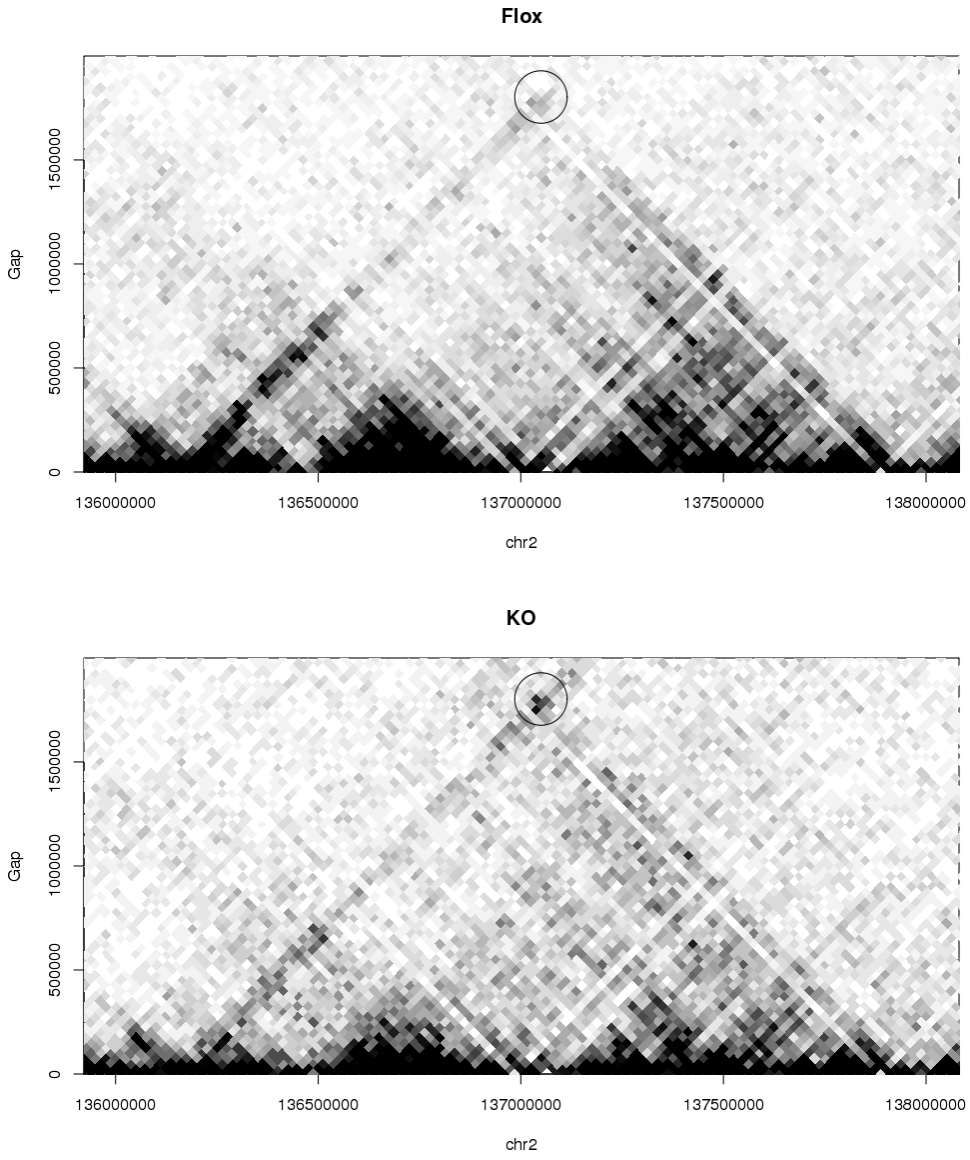
```
> chosen <- o.d[1]
> expanded.a <- resize(ax.d[chosen], fix="center", width=bin.size*5)
> expanded.t <- resize(tx.d[chosen], fix="center", width=bin.size*5)
> bound1 <- 30
> bound3 <- bound1*data$totals[3]/data$totals[1]
> plotPlaid(input[1], first=expanded.a, second=expanded.t, max.count=bound1,
+     width=5e4, param=mm.param, main="Flox")
> rect(start(ax.d[chosen]), start(tx.d[chosen]), end(ax.d[chosen]), end(tx.d[chosen]))
> plotPlaid(input[3], first=expanded.a, second=expanded.t, max.count=bound3,
+     width=5e4, param=mm.param, main="KO")
> rect(start(ax.d[chosen]), start(tx.d[chosen]), end(ax.d[chosen]), end(tx.d[chosen]))
```



7.3.2 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.

```
> chosen <- o.s[3]
> example <- tx.s[chosen]
> end(example) <- end(ax.s[chosen])
> nest.mid.a <- (results.n$nest.anchor.start[chosen]+results.n$nest.anchor.end[chosen])/2
> nest.mid.t <- (results.n$nest.target.start[chosen]+results.n$nest.target.end[chosen])/2
> nest.mid <- (nest.mid.a + nest.mid.t)/2
> nest.gap <- nest.mid.a - nest.mid.t
> rotPlaid(input[1], mm.param, region=example, width=2.5e4,
+     main="Flox", max.count=bound1)
> points(nest.mid, nest.gap, cex=7)
> rotPlaid(input[3], mm.param, region=example, width=2.5e4,
+     main="KO", max.count=bound3)
> points(nest.mid, nest.gap, cex=7)
```



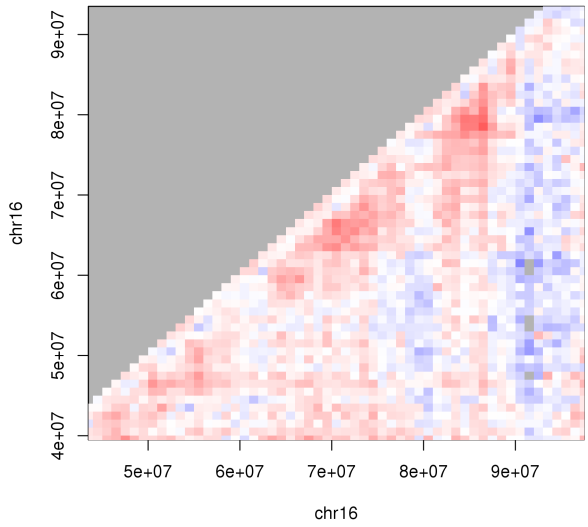
The circle marks the area of the interaction space that corresponds to the top-ranked nested bin pair within the `chosen` larger bin pair. An increase in the interaction intensity is clearly observed in the KO condition. This sharp change would not be observed with larger bin pairs, where the final count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

7.3.3 Using differential plaid plots

In some cases, it may be more informative to display the magnitude of the changes across the interaction space. This can be 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.

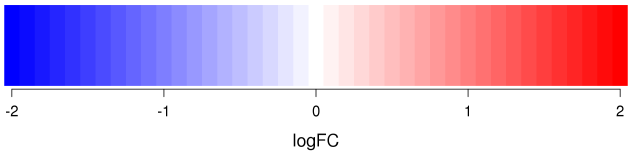
```
> chosen <- o.d[5]
> expanded.a <- resize(ax.d[chosen], fix="center", width=5e7)
> expanded.t <- resize(tx.d[chosen], fix="center", width=5e7)
> colfun <- plotDI(data, result$table$logFC, expanded.a, expanded.t, diag=FALSE)
```



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

Conventional feature detection

This chapter is a bit odd, so there's not much required from previous chapters. In particular, we'll only need the `input` and `mm.param` objects from Chapter 3.

8.1 Overview

Even though the diffHic package focuses on detecting DIs, some routines are also implemented for conventional analyses of Hi-C data. This includes the `correctedContact` function for iterative correction [Imakaev et al., 2012], and the `enrichedPairs` function for peak calling [Rao et al., 2014]. The following chapter presents some more functions that are designed to identify compartments and domains in the interaction space. These methods tend to be rather *ad hoc* from a statistical perspective, so more care is required in interpretation.

8.2 Identifying open and closed compartments

Contact frequencies in the interaction space of higher organisms typically exhibit a checkerboard pattern. This indicates that all sites on the genome must belong in one of two compartments, where loci in the same compartment interact more frequently than those in different compartments [Lieberman-Aiden et al., 2009]. The "open" compartment is associated with genes and active transcription, while the "closed" compartment is gene-poor. Identification of the loci in each compartment can be achieved with the `compartmentalize` function. In the example below, compartments are identified using bins at 500 kbp resolution.

```
> data <- squareCounts(input, width=5e5, filter=1, param=mm.param)
> output <- compartmentalize(data, centers=2)
```

Briefly, the `compartmentalize` function constructs a symmetric matrix of contact frequencies for each chromosome. Each row or column of the matrix represents a bin, and each entry

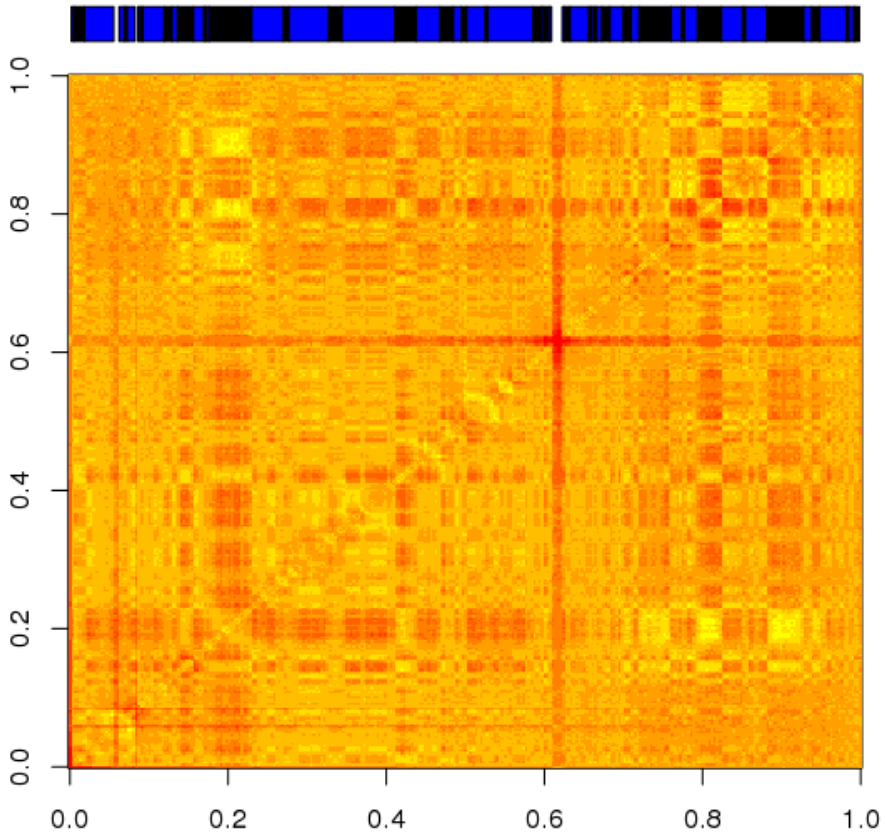
of the matrix represents the contact frequency – in this case, the average abundance across `input` libraries. Frequencies are normalized to remove distance-dependent trends and coverage biases between bins. The normalized matrix is then used in k-means clustering [Lajoie et al., 2015], where the bins (rows) are clustered based on similarities in their interaction profiles. All bins in the same cluster belong to the same compartment, as indicated by the cluster identifier in the returned `compartment` vector for each chromosome. The name of each entry in the vector refers to the index of `regions(data)` corresponding to that bin. Bins with `NA` entries correspond to low-coverage centromeres or telomeres that are filtered out.

```
> head(output$chr2$compartment, 20)

387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
 NA   1   1   2   2   2   2   2   1   1   1   1   1   1   1   1   1   1   1   1
```

By default, two clusters are identified in each chromosome that correspond to the open and closed compartments. More clusters can be specified by increasing `centers`, though the interpretation of the resulting compartments becomes more difficult. The function also returns the contact `matrix` for each chromosome. Users can then apply their own clustering algorithms on this matrix, if k-means is not satisfactory. Alternatively, the contact matrix can be used for visualization with `image`, as shown below for chromosome 5. The red/blue bar at the top of the plot indicates the genomic intervals in either compartment.

```
> par(xpd=TRUE)
> color.range <- c(-5, 5)
> temp <- output$chr5$matrix
> temp[temp < color.range[1]] <- color.range[1]
> temp[temp > color.range[2]] <- color.range[2]
> image(temp, zlim=color.range)
> current <- output$chr5$compartment
> stretches <- rle(current)
> ends <- cumsum(stretches$lengths) - 0.5
> starts <- ends - stretches$lengths
> not.na <- !is.na(stretches$values)
> rect(starts[not.na]/(nrow(temp)-1), 1.05, ends[not.na]/(nrow(temp)-1), 1.1,
+    col=c("black", "blue", "green")[stretches$values[not.na]])
```



Compartments can be identified at higher spatial resolution by using a lower `width` in `squareCounts`. To avoid running out of memory, users should `reform` the parameter object to `restrict` counting to intra-chromosomal bin pairs. However, be aware that small counts may compromise the effectiveness of the normalization scheme. High resolution may also be unnecessary given that compartments are usually quite large, e.g., 1 - 10 Mbp in size.

8.3 Identifying topological and other domains

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 [Nora et al., 2013]. At higher resolutions, small domains may also be formed due to looping between specific genomic elements [Rao et al., 2014].

To identify these domains, Dixon et al. [2012] introduced a directionality statistic for each genomic locus. 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 statistics at the start of the domain, followed by negative values at the end. These patterns can be extracted to identify the domain boundaries.

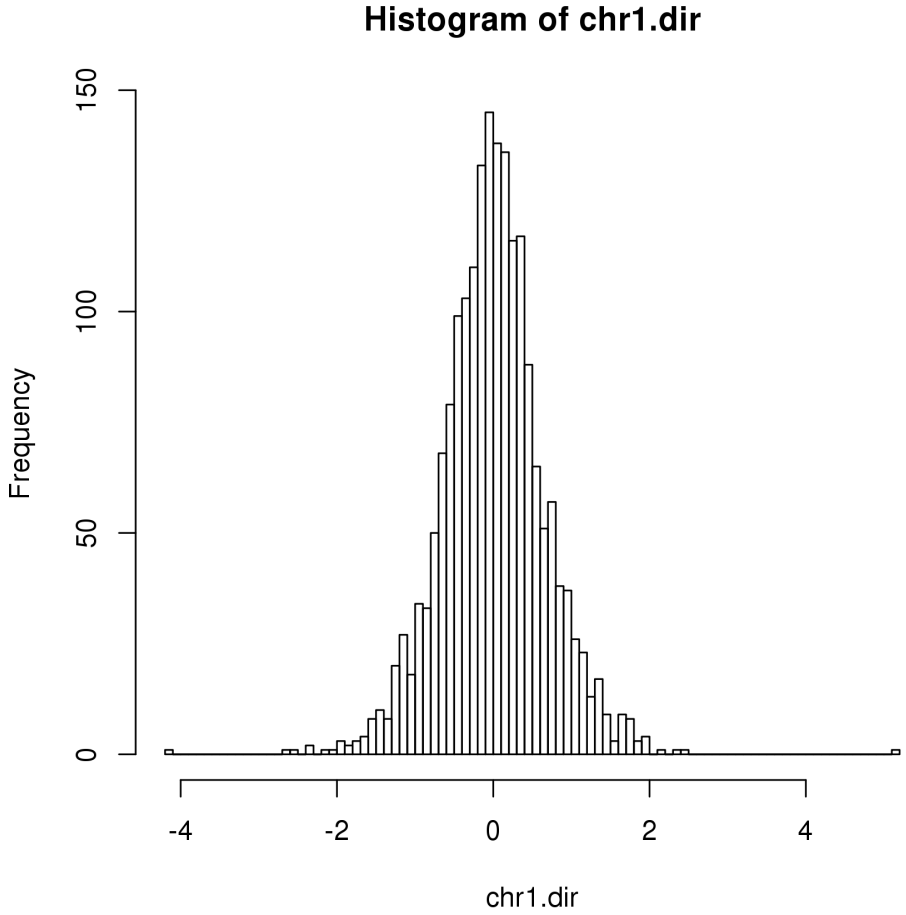
Directionality calculations are performed using the output of the `domainDirections` function. In the example below, the genome is first partitioned into 100 kbp bins. For each bin, the total number of read pairs mapped between that bin and its closest `span` upstream neighbors is counted. This is repeated for its closest `span` downstream neighbors. The genomic coordinates of each bin is returned along with the total count in either direction. If multiple libraries are specified in `input`, the average of the total counts is returned.

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

GRanges object with 26729 ranges and 3 metadata columns:
                seqnames              ranges strand   |    nfrags
                   <Rle>           <IRanges>  <Rle>   | <integer>
      [1]           chr1 [      1, 3004109]      *   |         1
      [2]           chr1 [3004106, 3100835]      *   |        34
      [3]           chr1 [3100832, 3194461]      *   |        30
      [4]           chr1 [3194458, 3301641]      *   |        41
      [5]           chr1 [3301638, 3399158]      *   |        52
      ...            ...                 ...    ... ...       ...
  [26725] chrUn_GL456393   [     1,  55711]      *   |         4
  [26726] chrUn_GL456394   [     1,  24323]      *   |         7
  [26727] chrUn_GL456396   [     1,  21240]      *   |         4
  [26728] chrUn_JH584304   [     1, 104610]      *   |        24
  [26729] chrUn_JH584304   [104607, 114452]      *   |         1
                       Up             Down
                 <numeric>        <numeric>
      [1] 44.9072835755743                0
      [2]  1153.7903395063 23.6121406225251
      [3] 839.361014665285 202.885540336862
      [4] 1491.96297784165 398.783690086507
      [5]  1799.7091232015 797.048016491232
      ...              ...              ...
  [26725]                0                0
  [26726]                0                0
  [26727]                0                0
  [26728]                0                0
  [26729]                0                0
  -------
  seqinfo: 66 sequences from an unspecified genome
```

The total counts can be used to calculate the directionality index defined by Dixon et al. However, a simpler alternative is to compute the log-fold change between the up/down counts for each bin. A large prior count is used to avoid spuriously large changes at low counts. The computed values seem to follow a Gaussian distribution, as shown for chromosome 1.

```
> prior.c <- 10
> directionality <- log2((finder$Up + prior.c)/(finder$Down + prior.c))
> onchr1 <- as.logical(seqnames(finder)=="chr1")
> chr1.dir <- directionality[onchr1]
> hist(chr1.dir, breaks=100)
```



Domain identification involves fitting a hidden Markov model (HMM) to the directionalities across each chromosome. Three states are used – upstream bias, no bias and downstream bias, corresponding to the start, middle and end of a domain with positive, near-zero and negative directionalities, respectively. The aim is to identify the most likely state for each

bin in the fitted model. HMM fitting can be performed in R using the depmixS4 package, as shown below. Note that the normality of the log-fold changes allows simple modelling with a single Gaussian distribution, instead the mixture of Gaussians used by Dixon et al.

```
> require(depmixS4)
> mod <- depmix(chr1.dir~1, family=gaussian(), nstates=3,
+    ntimes=sum(onchr1), trstart = runif(9))
> fm <- fit(mod, emc=em.control(rand=FALSE), verbose=FALSE)
```

Examination of the response parameters suggests that state 1 represents upstream bias, state 2 represents downstream bias and state 3 represents no bias. The transition probabilities also indicate that a direct transition from upstream to downstream states is unlikely, as is the downstream to no bias transition. This is consistent with the expected model of "upstream bias → no bias → downstream bias" state transitions for each domain.

```
> summary(fm)

Initial state probabilties model
pr1 pr2 pr3
  1   0   0


Transition matrix
               toS1         toS2         toS3
fromS1 7.286879e-01 1.336788e-06 2.713108e-01
fromS2 2.828607e-01 7.171393e-01 8.624896e-11
fromS3 9.436352e-08 1.760013e-01 8.239986e-01

Response parameters
Resp 1 : gaussian
    Re1.(Intercept)    Re1.sd
St1     0.651285032 0.5440686
St2    -0.654282230 0.5210010
St3    -0.008543609 0.2944410
```

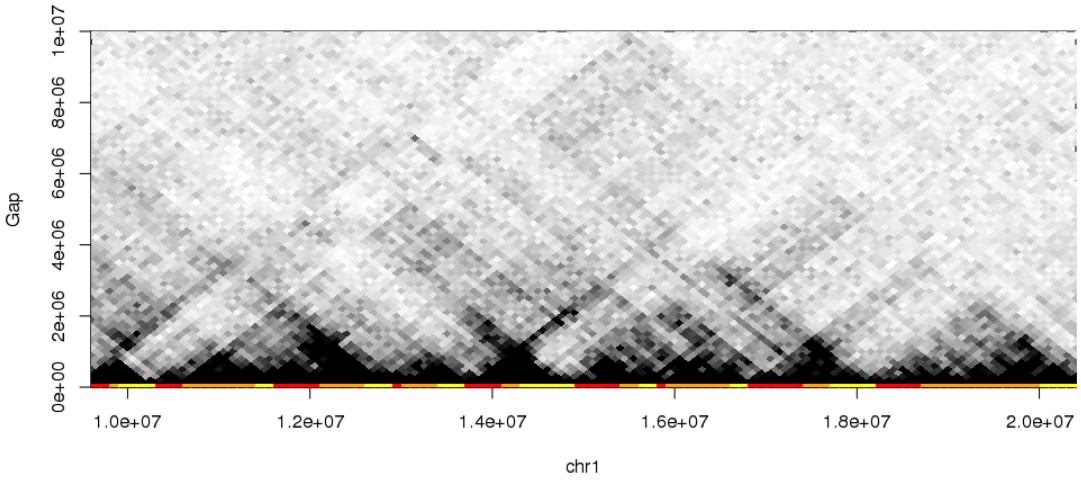
The most likely state for each bin can be determined by extracting the Viterbi path. Each pattern of "up → none → down" represents the interval of a single domain.

```
> all.states <- posterior(fm)$state
> head(all.states, n=50)

 [1] 1 1 1 1 1 1 1 3 3 3 3 2 2 2 2 2 1 1 1 1 1 3 3 3 3 3 3 3 2 2 2 2 2 1 1 1 1 3
[39] 3 2 2 2 2 2 2 1 3 3 3 3
```

The performance of this approach can be visually assessed with a (rotated) plaid plot. The red, yellow and orange intervals mark the start, end and center of each domain, respectively. These match up well with the characteristic triangle patterns in the interaction space.

```
> colors <- c("red", "yellow", "orange")
> rotPlaid(input[3], mm.param, GRanges("chr1", IRanges(10e6, 20e6)),
+     width=1e5, max.count=50)
> rect(start(finder[onchr1]), 0, end(finder[onchr1]), 1e5,
+    col=colors[all.states], border=NA)
```



As an aside, nesting of domains is often observed in real data [Filippova et al., 2014]. The HMM approach will only report a single set of small domains, and will not report the larger domains in which those small domains are nested. Additional effort will be required to identify the nesting structure. One possibility is to compute the total contacts between the reported domains, and to use this as a similarity measure for heirarchical clustering.

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 Lieberman-Aiden et al. dataset was obtained using the GEO accession number GSE18199. The Sofueva et al. dataset was obtained using the GEO accession GSE49017. Finally, the Rickman et al. dataset was obtained with 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.

Scripts for alignment and processing of these libraries can be accessed with the commands below. These scripts assume that the relevant Sequence Read Archive files have been downloaded. Some software packages are also required - read `sra2bam.sh` for more details.

```
> system.file('doc', 'sra2bam.sh', package="diffHic")
> system.file('doc', 'bam2hdf.R', package="diffHic")
```

9.2 Session information

```
> sessionInfo()
```

```
R version 3.2.0 (2015-04-16)
Platform: x86_64-unknown-linux-gnu (64-bit)
Running under: CentOS release 6.4 (Final)

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] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] depmixS4_1.3-2
 [2] Rsolnp_1.15
 [3] truncnorm_1.0-7
 [4] MASS_7.3-43
 [5] nnet_7.3-10
 [6] TxDb.Mmusculus.UCSC.mm10.knownGene_3.1.3
 [7] GenomicFeatures_1.21.16
 [8] AnnotationDbi_1.31.17
 [9] Biobase_2.29.1
[10] BSgenome.Mmusculus.UCSC.mm10_1.4.0
[11] BSgenome.Hsapiens.UCSC.hg19_1.4.0
[12] BSgenome_1.37.4
[13] rtracklayer_1.29.16
[14] Biostrings_2.37.5
[15] XVector_0.9.1
[16] edgeR_3.11.7
[17] limma_3.25.15
[18] diffHic_1.1.17
[19] GenomicRanges_1.21.18
[20] GenomeInfoDb_1.5.12
[21] IRanges_2.3.18
[22] S4Vectors_0.7.13
[23] BiocGenerics_0.15.6

loaded via a namespace (and not attached):
 [1] futile.logger_1.4.1        bitops_1.0-6
 [3] futile.options_1.0.0       tools_3.2.0
 [5] zlibbioc_1.15.0            biomaRt_2.25.1
 [7] statmod_1.4.21             RSQLite_1.0.0
 [9] rhdf5_2.13.5               lattice_0.20-33
[11] csaw_1.3.13                DBI_0.3.1
[13] locfit_1.5-9.1             grid_3.2.0
```

```
[15] XML_3.98-1.3               BiocParallel_1.3.48
[17] lambda.r_1.1.7             Rsamtools_1.21.15
[19] GenomicAlignments_1.5.12   splines_3.2.0
[21] SummarizedExperiment_0.3.3 KernSmooth_2.23-15
[23] RCurl_1.95-4.7
```

9.3 References

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.

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.

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

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

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.

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.

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

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.

D. Filippova, R. Patro, G. Duggal, and C. Kingsford. Identification of alternative topological domains in chromatin. *Algorithms Mol. Biol.*, 9:14, 2014.

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.

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.

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.

B. R. Lajoie, J. Dekker, and N. Kaplan. The Hitchhiker's guide to Hi-C analysis: practical guidelines. *Methods*, 72:65–75, Jan 2015.

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

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

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.

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.

C. Loader. *Local Regression and Likelihood*. Statistics and Computing. Springer New York, 1999. ISBN 9780387987750. URL `http://books.google.com.au/books?id=D7GgBAfL4ngC`.

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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

M. D. Robinson, 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.

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.

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

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.