diffHic: Differential analysis of Hi-C data

User's Guide

Aaron Lun

First edition 12 December 2012

Last revised 20 January 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(preparePET)` at the R prompt. Otherwise, try reading this guide or contacting one of the authors for more information. Considered suggestions for improvements are occasionally appreciated.

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 favoured. The ligated DNA is sonicated and the fragments of DNA containing ligation junctions are purified by a biotin pulldown. The ligation product is then subjected to paired-end sequencing. Mapping of the reads in each pair can identify

the 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 <- normalize(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 smaller 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

Hello, dear reader. A little box will appear at the start of each chapter, describing the objects that are required from the previous chapter. As we're starting out here, we don't really need anything except your enthusiasm.

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. Correct alignment of the 5′ end of the read is more important. This is because the location of the 3′ end is already provided by the location of 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.

Alignment of chimeric reads can be achieved 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://picard.sourceforge.net`).

2.2 Matching mapped reads to restriction fragments

The Hi-C protocol is based on ligation between restriction fragments, i.e., the genomic interval between adjacent restriction sites [Lieberman-Aiden et al., 2009]. 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 will automatically check the validity of the ordering of the fragments. The object can also hold several other parameters for use in read pair counting, later. 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 can be used to match the mapping location of each read to a restriction fragment. Mapping results for each read in each pair are provided as a name-sorted BAM file. The function then converts the read position into the index of the corresponding entry 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.

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

$pairs
   total   marked filtered   mapped
 7068675   103547  1657440  5337807

$same.id
   dangling self.circle
     423211      135008

$singles
[1] 0

$chimeras
  total  mapped   multi invalid
2297481 1656730 1072926   69482
```

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. Dangling ends are inward-facing read pairs that are mapped to the same fragment [Belton et al., 2012]. 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 downstream confusion.

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 align-

ment, 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 just work normally if the mapping location is only given for the 5′ end, e.g., as with iterative mapping.

The proportion of invalid chimeric pairs can also 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 of the error rate, given that split alignments are less reliable than those using the full read sequence. Invalid chimeric pairs can be discarded by setting `ichim=FALSE` in `preparePairs`. However, this is not recommended as mapping errors for short 3′ ends may result in 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   858203    93666    82131  3758082
```

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.

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) missed by `preparePairs`, as the two reads involved are mapped to different restriction fragments.

2.4.2 Setting parameter values with strand orientation plots

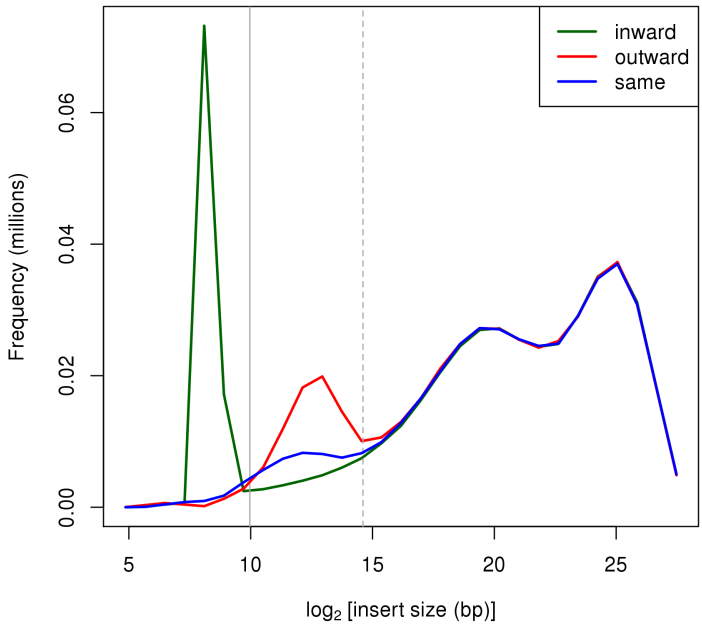
The strand orientation for a read pair refers to the combination of strands for the anchor/target reads. These are stored as flags where setting `0x1` or `0x2` means that the anchor or target reads, respectively, are mapped on the reverse strand. If different pieces of DNA were randomly ligated together, 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 with `getPairData`. Each orientation is equally represented, which is expected as different chromosomes represent different pieces of DNA.

```
> diags <- getPairData("SRR027957.h5", hs.param)
> intra <- !is.na(diags$insert)
> table(diags$orientation[!intra])

     0      1      2      3
750007 746596 746616 746497
```

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. These are not removed by `preparePairs` as the read pairs are not mapped to the same restriction fragment. Rather, the artifacts are formed from incompletely digested fragments. 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)
```



It is possible that these spikes reflect some genuine aspect of chromatin organization. For example, the spike in the outward-facing read pairs may be the result of systematic outward looping in chromatin packaging. Removal of the spikes would preclude the detection of such features. Nonetheless, filtering is still recommended to prevent genuine interactions from being dominated by technical artifacts. Indeed, even if there are imbalances in the strand orientations for any one genuine interaction, these skews should balance out when read pairs from all interactions are considered (i.e., systematic preferences assumed to be absent).

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 compatability, users should include the alignment positions and lengths as `xxx.pos` and `xxx.len` (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 should be stored for both the anchor and target alignments. It is more useful for other diffHic functions than some arbitrary `other.data`.

Chapter 3

Counting read pairs into interactions

A different dataset is used in this chapter, so we really don't need anything from the previous chapter. Just stay enthused, because this is where the fun starts.

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. This must be 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 read pairs are distributed. This refers to the space of 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. A rectangular area in the interaction space represents a pairwise interaction between the genomic intervals spanned by the two adjacent sides of the rectangle. The number of read pairs in each area is then 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 3319104 pairs across 2739 regions

Counts:
      [,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  8984  8443
... and 3319099 more rows

Totals:
[1] 85786442 74685222 60860615 54596195

Anchors:
   Chr   Start     End
1 chr1 3004106 4000741
2 chr1 3004106 4000741
3 chr1 4000738 5001375
4 chr1 4000738 5001375
5 chr1 4000738 5001375
... and 3319099 more rows

Targets:
   Chr   Start     End
1 chr1       1 3004109
2 chr1 3004106 4000741
3 chr1       1 3004109
4 chr1 3004106 4000741
5 chr1 4000738 5001375
... and 3319099 more rows
```

This generates a `DIList` object containing the relevant information. Each row of the count matrix represents the counts for an interaction, while each column represents a library. Each

interaction is characterized as a pair of genomic intervals, i.e., bins. Again, anchor and target notation is used for these intervals, whereby the anchor bin is that with the higher genomic coordinate. The total vector just contains the total number of read pairs in each library.

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 with 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.

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 7.2.4 for more information on consolidating results from multiple bin sizes.

```
> 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.

3.3 Counting with pre-defined 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(org.Mm.eg.db)
> gene.data <- select(org.Mm.eg.db, keys=keys(org.Mm.eg.db),
+     columns=c("CHRLOC", "CHRLOCEND"), keytype="ENTREZID")
> gene.data <- gene.data[!is.na(gene.data$CHRLOCCHR),]
> gene.body <- GRanges(paste0("chr", gene.data$CHRLOCCHR),
+     IRanges(abs(gene.data$CHRLOC), abs(gene.data$CHRLOCEND)))
```

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 can 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 16265768 pairs across 27292 regions

Counts:
     [,1] [,2] [,3] [,4]
[1,] 9540 7469 5967 5739
[2,]  701  604  329  334
[3,] 1182  933  835  799
[4,]  127  134   68   52
[5,]  424  315  278  274
... and 16265763 more rows

Totals:
[1] 85786442 74685222 60860615 54596195

Anchors:
   Chr   Start     End
1 chr1 3211067 3675240
2 chr1 4286646 4409818
3 chr1 4286646 4409818
4 chr1 4340420 4362532
5 chr1 4340420 4362532
... and 16265763 more rows

Targets:
   Chr   Start     End
1 chr1 3211067 3675240
2 chr1 3211067 3675240
3 chr1 4286646 4409818
4 chr1 3211067 3675240
5 chr1 4286646 4409818
... and 16265763 more rows
```

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 2 metadata columns:
      seqnames             ranges strand |    nfrags  original
         <Rle>          <IRanges>  <Rle> | <integer> <integer>
  [1]     chr1 [3211067, 3675240]      * |       197     24086
  [2]     chr1 [4286646, 4409818]      * |        49      4725
  [3]     chr1 [4340420, 4362532]      * |         8      4724
  [4]     chr1 [4487488, 4498343]      * |         2      5193
  [5]     chr1 [4772601, 4786029]      * |         3      6667
  [6]     chr1 [4802092, 4847111]      * |         8      4156
  -------
  seqinfo: 35 sequences from an unspecified genome; no seqlengths
```

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 single large gene body will not be resolved.

3.4 Counting into single bins

For each bin, the number of read pairs with at least one read mapped inside that bin can be counted with the `marginalCounts` function. This effectively uses the Hi-C data to examine 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 the differential analysis.

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

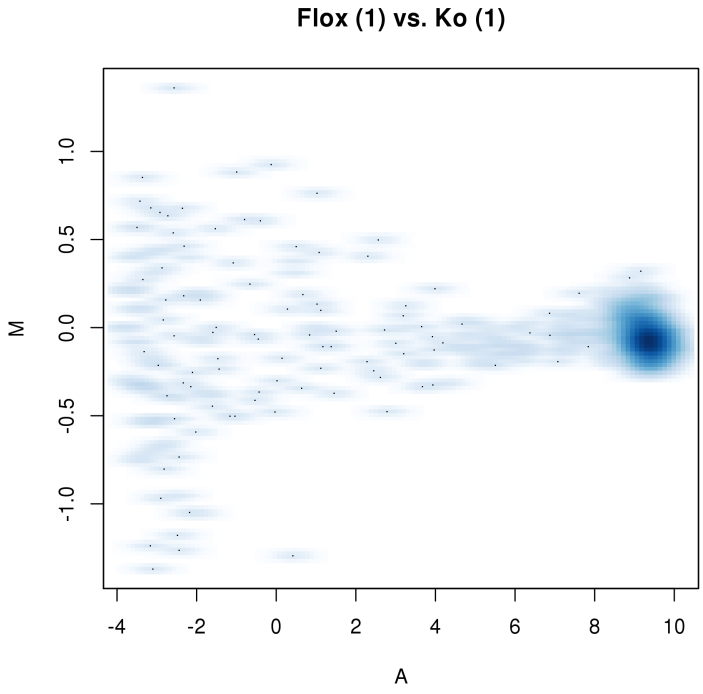
```
DIList object for 4 libraries with 2732 pairs across 2739 regions

Counts:
      [,1]  [,2]  [,3]  [,4]
[1,]   203   128   106    82
[2,] 78448 65876 57030 54474
[3,] 72665 63262 52369 48886
```

```
[4,] 68766 59567 49858 46784
[5,] 70338 62625 53813 50920
... and 2727 more rows

Totals:
[1] 85786442 74685222 60860615 54596195

Anchors:
   Chr   Start     End
1 chr1       1 3004109
2 chr1 3004106 4000741
3 chr1 4000738 5001375
4 chr1 5001372 5997485
5 chr1 5997482 7000260
... and 2727 more rows

Targets:
   Chr   Start     End
1 chr1       1 3004109
2 chr1 3004106 4000741
3 chr1 4000738 5001375
4 chr1 5001372 5997485
5 chr1 5997482 7000260
... and 2727 more rows
```

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 5.3.

```
> 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)")
```



It must be stressed that the marginal count refers to a count for each bin. To avoid any confusion, the count for each *bin pair* will be described as the interaction count. Later, it may be necessary to adjust the interaction count based on the marginal counts for the corresponding bins (Section 5.3, for example). In such cases, one should ensure that the same values of `width` and `fragments` are used in both `squareCounts` and `marginCounts`. One can check that this is the case by ensuring that the regions are the same.

```
> identical(regions(data), regions(margin.data))
```

```
[1] TRUE
```

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 1-by-2 matrix, count loading will be limited to the read pairs that are mapped between the specified pair 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. Reads from several repeat units in a real genome may be collated into a single representative unit in the genome build. This results in a sharp spike in read intensity, incorrectly suggesting a sharp feature in the interaction space. The problem is exacerbated by different repeat copy numbers between 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 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 interaction quantification. 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 change the values if necessary, to 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 are 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.frag`. Enthusiasm is desirable, but no longer necessary.

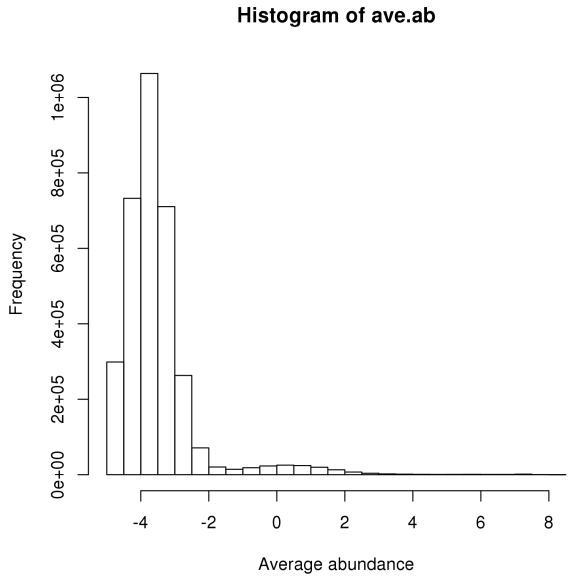
4.1 Overview

4.1.1 Computing the average abundance in a NB model

Filtering can remove uninteresting features in analyses of high-throughput experiments. This reduces the severity of the multiple testing correction and increases detection 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. Here, the overall NB mean across all libraries is (probably) an independent filter statistic. This is also called the average abundance and can be computed using the `aveLogCPM` function in edgeR [McCarthy et al., 2012], as shown below.

```
> 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. Low counts will also interfere with the asymptotic approximations that will be used in downstream statistical modelling. The example below removes those bin pairs with an overall NB mean below 5 across all libraries.

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

   Mode   FALSE    TRUE    NA's
logical 2433381  885723       0

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

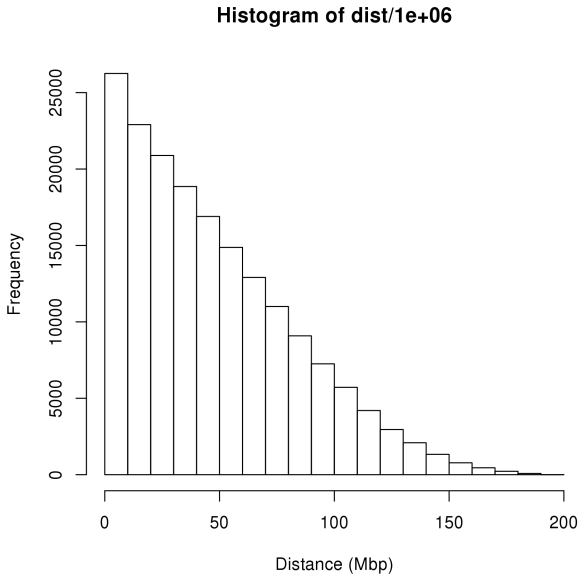
This count-based approach is fairly objective yet is still effective, i.e., removes a large number of bin pairs. More sophisticated strategies can be implemented where the choice of threshold is motivated by some understanding of the Hi-C protocol. These strategies are described in the rest of this chapter, and will be combined with the count-based filter to maintain a minimum count size among the retained bin pairs.

Here, the filtered results are assigned into the `dummy` object. This avoids overwriting the original data at this point in the guide. Similarly, `dummy` will be used as a dumping ground for the various demonstrations below. However, when actual filtering is desired, the filtered results should be assigned back to the `data` object for further analysis.

4.1.2 Computing the interaction distance

The linear distance between each pair of the interacting bins is worth mentioning as it can also be used in filtering. This is obtained using the `getDistance` function, which computes the distance between the bin midpoints by default.

```
> dist <- getDistance(data, type="mid")
> hist(dist/1e6, xlab="Distance (Mbp)")
```



Inter-chromosomal bin pairs are marked here as `NA` for completeness. 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.

```
> summary(is.na(dist))

   Mode   FALSE    TRUE    NA's
logical  178758 3140346       0
```

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 interaction is present to drive ligation between the corresponding bins. Any changes in the counts for

these bin pairs are not interesting 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. The threshold is defined by requiring a minimum fold change of 10 above this estimate.

```
> direct.threshold <- median(ave.ab[is.na(dist)], na.rm=TRUE)
> direct.threshold

[1] -3.72137

> direct.keep <- count.keep & ave.ab > log2(10) + direct.threshold
> dummy <- data[direct.keep,]
> summary(direct.keep)

   Mode   FALSE    TRUE    NA's
logical 3197231  121873       0
```

The `direct.keep` vector can then be applied for filtering of `data`, as previously shown with `count.keep`. This approach is named here as "direct" filtering, as the average count is directly compared against a fixed threshold value. Note that the construction of `direct.keep` is based off `count.keep` to ensure that retained bins have large absolute counts.

4.3 Filtering as a function of interaction distance

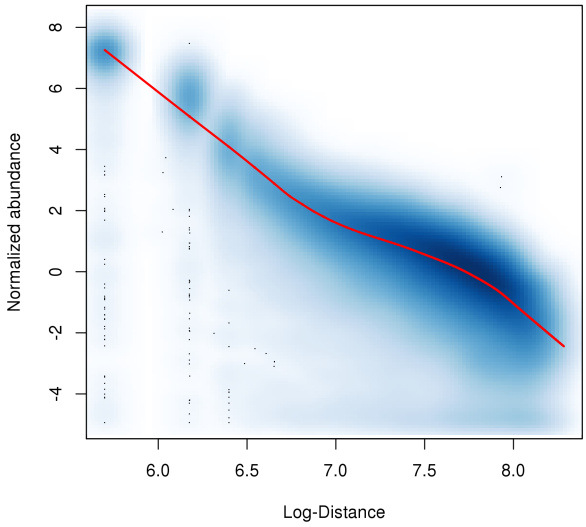
A more complex filter adjusts the threshold according to the distance between the bins in each bin pair. Larger counts are observed at lower distances, suggesting that a concomitantly higher threshold is necessary. This expands the definition of "uninteresting" events to include those interactions that are formed by simple compaction of chromatin [Lin et al., 2012].

In this strategy, a trend is fitted to the average abundance for all intra-chromosomal bin pairs using the log-distance as the covariate. Half of the bin size is added to the distance as a prior, to avoid undefined values when distances are zero. Inter-chromosomal bin pairs will not be involved in trend fitting as the distance will be `NA`. Nonetheless, the fitted value for these bin pairs is set to the fitted value of the largest intra-chromosomal distance.

```
> log.dist <- log10(dist + bin.size/2)
> trend.threshold <- loessFit(x=log.dist, y=ave.ab)$fitted
> trend.threshold[is.na(log.dist)] <- trend.threshold[which.max(log.dist)]
```

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 typically 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(log.dist, ave.ab, xlab="Log-Distance", ylab="Normalized abundance")
> o <- order(log.dist)
> lines(log.dist[o], trend.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., the intensity is above that expected from compaction. This favours selection of bin pairs corresponding to long-range interactions.

```
> trend.keep <- count.keep & ave.ab > trend.threshold
> dummy <- data[trend.keep,]
> summary(trend.keep)

   Mode    FALSE    TRUE    NA's
logical 3162548  156556       0
```

4.4 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, e.g., for small bin sizes where too many non-empty bin pairs are present. In such cases, two 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. This is quite slow.
- 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)
```

The `scaledAverage` function is applied to compute the average abundance for each of the small bin pairs. This adjusts for the area of the interaction space used for read pair counting, and is necessary for a valid comparison between small and large bin pairs. For consistency, the abundances of the larger bin pairs are also recomputed. The resulting values are used to obtain a direct filter threshold, as previously described.

```
> smaller.ab <- aveLogCPM(asDGEList(smaller.data))
> summary(smaller.ab)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -3.339  -3.041  -2.509  -1.931  -1.302   5.512

> require(csaw)
> larger.ab <- scaledAverage(asDGEList(data), scale=(bin.size/new.bin.size)^2)
> new.threshold <- median(larger.ab[is.na(dist)], na.rm=TRUE)
> new.threshold

[1] -5.085195
```

Finally, filtering is performed by mandating a minimum fold change for the abundances of the smaller bin pairs over the adjusted threshold (in this case, 100-fold). More bin pairs are retained here, as the total number of bin pairs is greater when the bin sizes are smaller. Also recall that a large `filter` is used in `squareCounts` to generate `smaller.data`. This means that no additional filtering is necessary to enforce a minimum count size.

```
> small.keep <- smaller.ab > new.threshold + log2(10)
> smaller.filtered <- smaller.data[small.keep,]
> summary(small.keep)

   Mode   FALSE    TRUE    NA's
logical  720114  340737       0
```

The same approach can be used for trended filtering of small bin pairs. However, some extra effort is required to match the threshold at each covariate. This is done by performing interpolation (and, if necessary, extrapolation) of the fitted distance-abundance trend for the distances of the smaller bin pairs. The computed `new.threshold2` can then be used to filter on the abundances of the smaller bin pairs, as described above.

```
> new.log.dist <- log10(getDistance(smaller.data) + new.bin.size/2)
> larger.fit <- loessFit(x=log.dist, y=larger.ab)$fitted
> new.threshold2 <- approx(x=log.dist, y=larger.fit, xout=new.log.dist, rule=2)$y
> new.threshold2[is.na(new.log.dist)] <- larger.fit[which.max(log.dist)]
> summary(smaller.ab > new.threshold2)

   Mode   FALSE    TRUE    NA's
logical  516591  544260       0
```

Another advantage is that estimation of the threshold is also more precise when the counts are larger. For example, calculation of the median or fitting of the trend will be complicated by discreteness in the abundances. This is avoided with larger bin sizes. Thus, even if there is sufficient memory for loading smaller bin pairs with `filter=1`, use of larger bin sizes may still be desirable when computing the filter threshold.

4.5 Filtering for pre-specified regions

Filtering for pairs of arbitrary regions is complicated by the potential irregularity of the regions. In particular, there is no guarantee that the supplied regions will cover the entirety of the interaction space. This means that the filter thresholds may be overestimated, e.g., if the regions were specified such that only high-abundance areas are counted. Similarly, threshold estimation may be imprecise if not enough areas are non-empty.

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)))
```

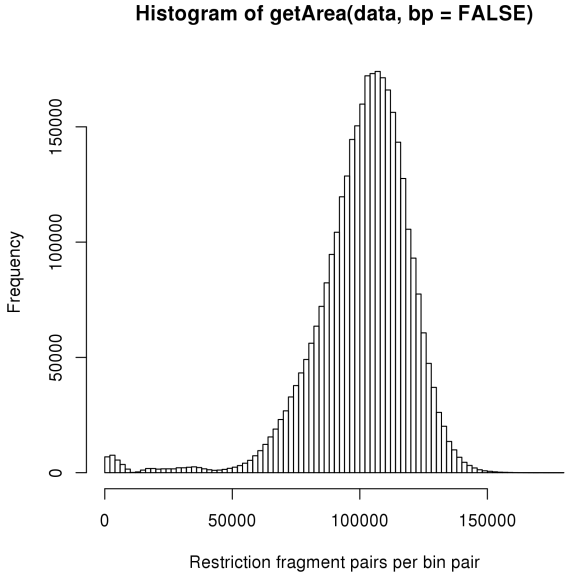
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-4.942  -4.939  -4.936  -4.762  -4.786   8.866
```

Nonetheless, other users may wish to adjust for the differences in areas. In such cases, `scaledAverage` can be used to compute the filter statistic in conjunction with `getAreas`. Computed areas are mean-adjusted and used to scale the average abundances for each region pair. Note that this differs from the application of `scaledAverage` in Section 4.4. There, the scaling factor is constant and does not alter the relative abundances within each set of bin pairs (i.e., of the same size). Here, each region pair has a different area such that its abundance will be subject to different scaling.

```
> area <- getArea(redata, bp=TRUE)
> area <- area/mean(area)
> summary(scaledAverage(asDGEList(redata), scale=area))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-5.108  -4.839  -4.352  -3.902  -3.430  10.820
```

On a similar note, another approach defines the area for each region/bin pair based on the number of restriction fragment pairs. This may be more relevant as it accounts for the limits of spatial resolution in a Hi-C experiment. For example, bins of similar size may have different counts if they contain different numbers of fragments. In practice, fragment-based area adjustment prior to filtering is not recommended. This is because it tends to select for uninteresting bins with very few fragments, e.g., telomeres, centromeres.

```
> hist(getArea(data, bp=FALSE), breaks=100, xlab="Restriction fragment pairs per bin pair")
```



In all cases, adjustment of abundances by area fails to account for other differences between bins, e.g., in mappability or sequenceability. Full consideration of these bin-specific biases requires more complex methods such as iterative correction (see Section 5.2).

4.6 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 behaviour 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 applying the `diag.keep` object alongside the direct or trended filters, as shown below.

```
> diag.keep <- is.na(dist) | dist > 0L
> dummy <- data[diag.keep & direct.keep,]
> summary(diag.keep)

   Mode   FALSE    TRUE    NA's
logical    2614 3316490       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. Artifacts will still be restricted to diagonal bin pairs, so long as the bins are larger than ∼25 kbp (see Section 2.4.2). Short-range interactions will then be represented by the remaining off-diagonal bin pairs, due to the improved spatial resolution of smaller bins. While count sizes will also decrease, this should not be a major problem as counts should be larger at low interaction distances.

4.7 Peak-calling in the interaction space

Filtering can also be performed to select for peaks in the interaction space. This refers to bin pairs with substantially more reads than its neighbours. The `enrichedGap` function computes an enrichment value for each bin pair, defined as the log-fold change of its abundance against its neighbours. Those bin pairs with high enrichment values can then be retained. Note that the function requires data without any filtering. Thus, to avoid overloading the machine memory, it may be prudent to analyze each pair of chromosomes separately.

```
> chr1.data <- squareCounts(input, reform(mm.param, restrict="chr1"), width=new.bin.size)
> enrich.chr1 <- enrichedGap(chr1.data, width=new.bin.size, flank=5)
> summary(enrich.chr1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-4.0230 -0.2753 -0.1086 -0.1092  0.0648  2.1040
```

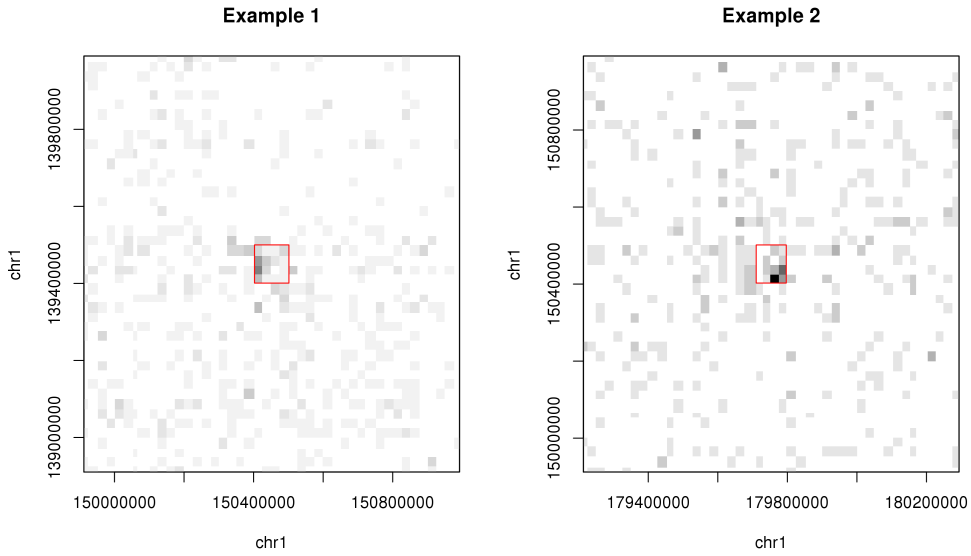
More specifically, the function partitions the neighbourhood into eight different regions around the target bin pair. The average abundance across each region is computed, and the region with the largest abundance is chosen. The enrichment value is defined as the log-fold change between the abundances of the target bin pair and the chosen region. These regions are designed to capture any high-intensity structural features like TADs, compartments or banding patterns [Rao et al., 2014] that might be present. Thus, any bin pair lying within one of these features will be compared to other high-abundance bin pairs from the same feature. Otherwise, peaks may be spuriously detected at the feature boundary, where the neighbourhood is diluted by low-abundance bin pairs outside of the feature.

Some adjustment can also be performed to account for distance biases. A loess curve is fitted to the average abundances against the log-distances, and the fitted value is subtracted from the abundance for each intra-chromosomal bin pair. This allows a valid comparison between bin pairs at different distances in the interaction space (assuming that the distance effect is uninteresting, e.g., due to compaction). Of course, this is not required for inter-chromosomal bin pairs, and can be turned off altogether by setting `trend="none"`.

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 surrounding neighbourhood is empty. The example below retains those bin pairs with average abundances and enrichment values above 1. Both thresholds are arbitrarily chosen as they ultimately depend on what the user considers to be interesting.

```
> chr1.ab <- aveLogCPM(asDGEList(chr1.data))
> keep <- enrich.chr1 > 1 & chr1.ab > 1
> sum(keep)

[1] 57
```

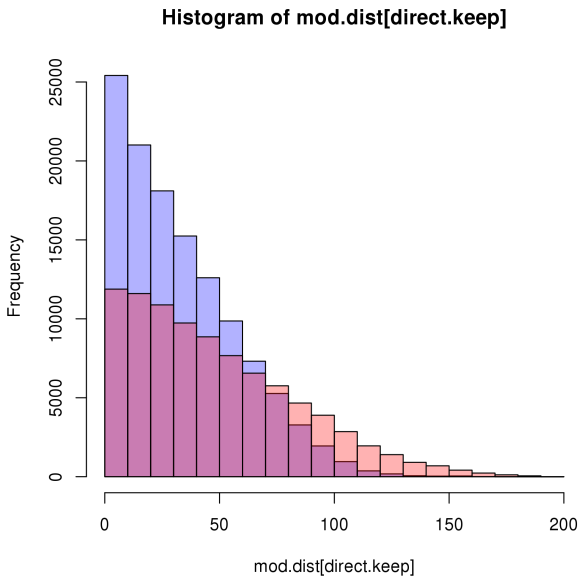
This filtering strategy can be evaluated by examining the interaction space around each putative peak (see Chapter 7.3). Briefly, the colour intensity of each "pixel" is proportional to the number of read pairs between the corresponding loci on each axis. Red boxes mark the bin pairs retained by filtering, where each side represents a bin. In both examples, the bin pairs correspond nicely to punctate high-intensity contacts in the interaction space.



4.8 Summary of the filtering strategies

Each filtering strategy is arbitrarily tunable, as one can simply increase or decrease the minimum fold change required for retention. Nonetheless, they 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 just arbitrarily chosen. The effect of each approach can also be summarized by examining the distribution of interaction distances after filtering.

```
> mod.dist <- dist/1e6
> all.breaks <- hist(mod.dist, plot=FALSE)$breaks
> hist(mod.dist[direct.keep], breaks=all.breaks, col=rgb(0,0,1,0.3))
> hist(mod.dist[trend.keep], breaks=all.breaks, col=rgb(1,0,0,0.3), add=TRUE)
```



As expected, the direct method preferentially retains short-range interactions whereas the trended method selects for long-range interactions. 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 retention of short-range interactions, given that short-range packaging dominates genomic organization. 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.keep,]
> smaller.data <- smaller.filtered
```

As promised, 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 5.2, but can otherwise be ignored. Direct filtering is also performed for the smaller bin pairs, and the results stored in `smaller.data`.

Chapter 5

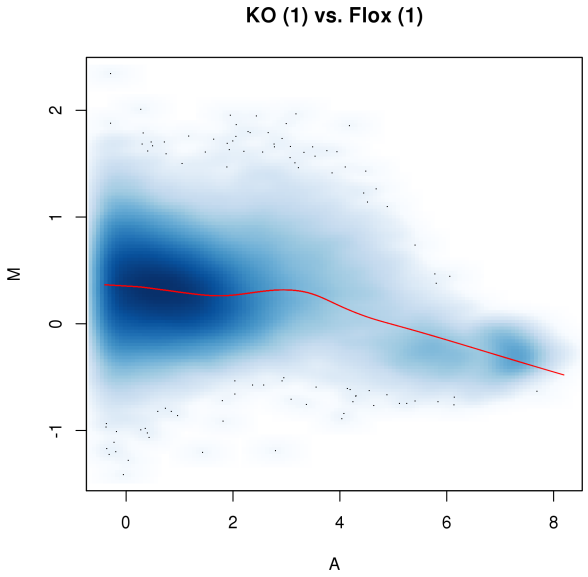
Normalization strategies for Hi-C data

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 Removing trended biases between libraries

Library-specific 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[,1]
> smoothScatter(ab, mval, xlab="A", ylab="M", main="KO (1) vs. Flox (1)")
> fit <- loessFit(x=ab, y=mval)
> lines(ab[o], fit$fitted[o], col="red")
```



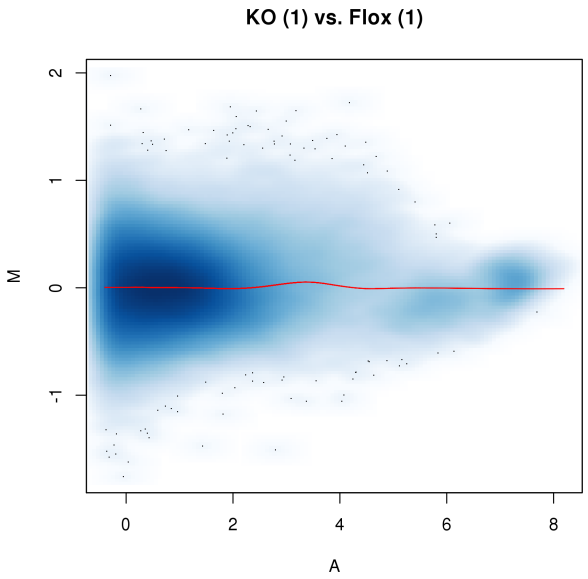
Trended biases are problematic as they can inflate the variance estimates or fold-changes for some bin pairs. Thus, they must be eliminated with non-linear normalization. The code below uses the NB-loess method, which is an adaptation of existing non-linear methods for discrete count data [**?**]. An offset term is computed for each bin pair in each library, for use in a generalized linear model (GLM). A large offset for an observation is equivalent to downscaling the corresponding count relative to the counts of the other libraries.

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

```
          [,1]      [,2]       [,3]       [,4]
[1,] 0.3879109 0.2098552 -0.2533659 -0.3444002
[2,] 0.3045463 0.1538210 -0.1852569 -0.2731104
[3,] 0.3580988 0.1899629 -0.2290995 -0.3189622
[4,] 0.2277857 0.1020684 -0.1225062 -0.2073479
[5,] 0.2952056 0.1474616 -0.1775803 -0.2650869
[6,] 0.3592615 0.1907406 -0.2300473 -0.3199548
```

The MA plot can be examined after adjusting the log-counts with the offsets. Most of the trend is removed which indicates that normalization was successful. Of course, this 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.

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



Filtering on average abundance prior to normalization 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.

5.2 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]         NA         NA 0.14653918         NA         NA 0.03264732
```

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 behaviour during correction.

Note that `original` is used as no filtering should be performed prior to `correctedContact`. All non-empty bin pairs are needed for correction as information is collected across the entire interaction space. The contribution of many bin pairs with low counts is as substantial as that of few bin pairs with large counts. The convergence of the correction procedure can then be checked by examining the maximum fold change to the truth at each iteration.

```
> corrected$max
```

```
 [1] 186.770783   8.028471   2.842165   1.703590   1.325088   1.171107
 [7]   1.099043   1.061208   1.039449   1.026089   1.017523   1.011879
[13]   1.008099   1.005541   1.003799   1.002608   1.001792   1.001232
[19]   1.000847   1.000583   1.000401   1.000276   1.000190   1.000130
[25]   1.000090   1.000062   1.000042   1.000029   1.000020   1.000014
[31]   1.000009   1.000007   1.000005   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
```

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.

5.3 Accounting for copy number variations

5.3.1 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, which simultaneously removes trended biases as well. The function is based on multi-dimensional smoothing across several covariates with the `locfit` package [Loader, 1999].

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

```
          [,1]      [,2]       [,3]       [,4]
[1,] 0.3998604 0.1590148 -0.2794132 -0.2794620
[2,] 0.3124102 0.1465652 -0.2205689 -0.2384065
[3,] 0.3707134 0.1838412 -0.2659823 -0.2885723
[4,] 0.2322682 0.0871152 -0.1529915 -0.1663919
[5,] 0.3032015 0.1426633 -0.2119763 -0.2338886
[6,] 0.3670021 0.1750749 -0.2618522 -0.2802248
```

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.

5.3.2 Why use multi-dimensional smoothing?

The use of a empirical fit reduces the number of assumptions involved in translating a CNV into its effect on interaction intensities. For example, iterative correction could correct for CNVs by accounting for differences in coverage of each bin. However, converting the change in coverage into a quantifiable change in the interaction intensity requires the assumption of factorizability [Imakaev et al., 2012], i.e., the effect on intensity is a product of the biases of the interacting regions. This is reasonable under a random ligation model, but does not account for other mechanisms of read pair generation.

The fit must be performed simultaneously with all covariates, to ensure that biases at any combination of covariate values are properly identified (and removed). That said, the use of many covariates may lead to overfitting and removal of genuine differences. The function also assumes that CNVs in different parts of the genome have the same effect on the interaction intensities. Caution is therefore required when using `normalizeCNV`. The safest choice is to avoid it when CNVs are not present, based on the diagnostic plots in Section 3.4.

As an aside, 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.

5.3.3 Visualizing the effect of CNV removal

Here, the Rickman et al. dataset is used for demonstration 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.

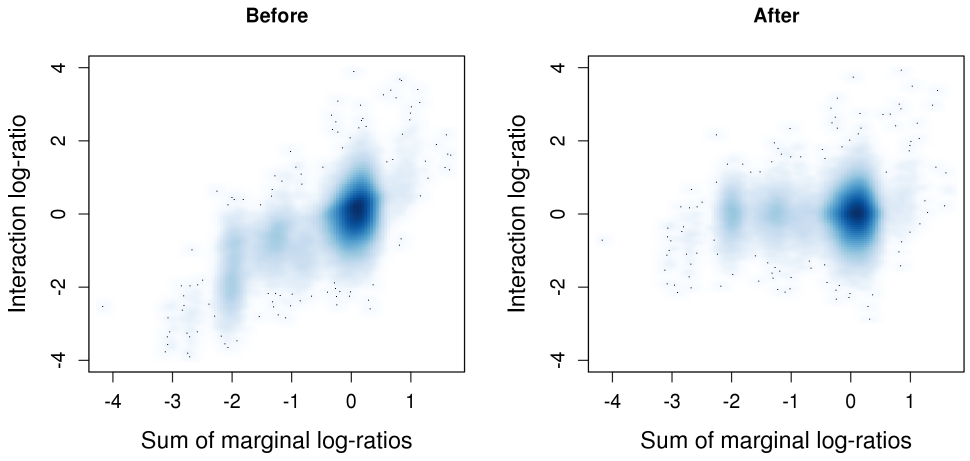
```
> count.files <- c("merged_erg.h5", "merged_gfp.h5")
> rick.data <- squareCounts(count.files, hs.param, width=1e6)
> rick.marg <- marginCounts(count.files, hs.param, width=1e6)
> rick.data <- rick.data[aveLogCPM(asDGEList(rick.data)) > 0,]
```

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 magnitude of biological variability can be empirically determined from biological replicates, i.e., Hi-C libraries prepared from different biological samples. This is used during testing to appropriately reduce the significance of any detected differences when the data is highly variable. For count-based data, this is achieved using the NB model in edgeR [Robinson et al., 2010]. Estimation of the NB dispersion parameter allows modelling of the variation between biological replicates. 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]. This means that a design matrix must be specified to describe the experimental setup. Here, a simple one-way layout is sufficient. The code below specifies two groups of two replicates, where each group corresponds to a single cell type. At this point, 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"
```

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
```

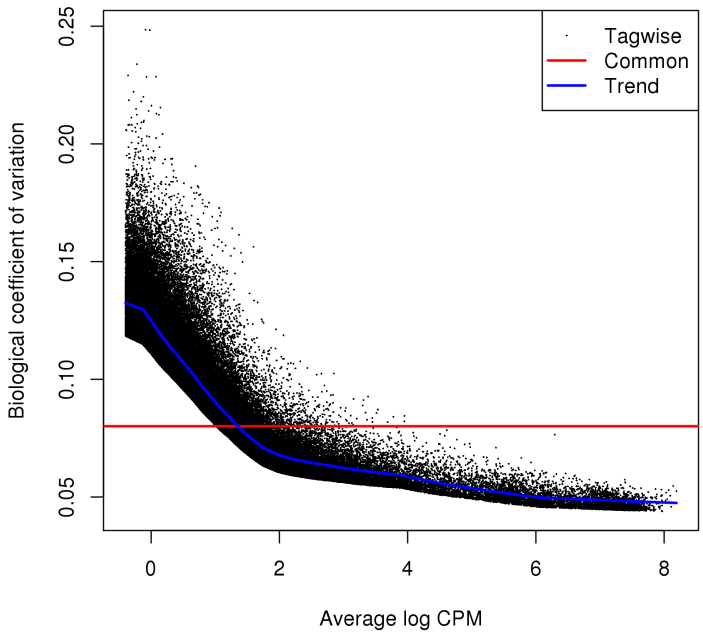
6.2 Estimating the NB dispersion

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] 0.006412909
```

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)
```



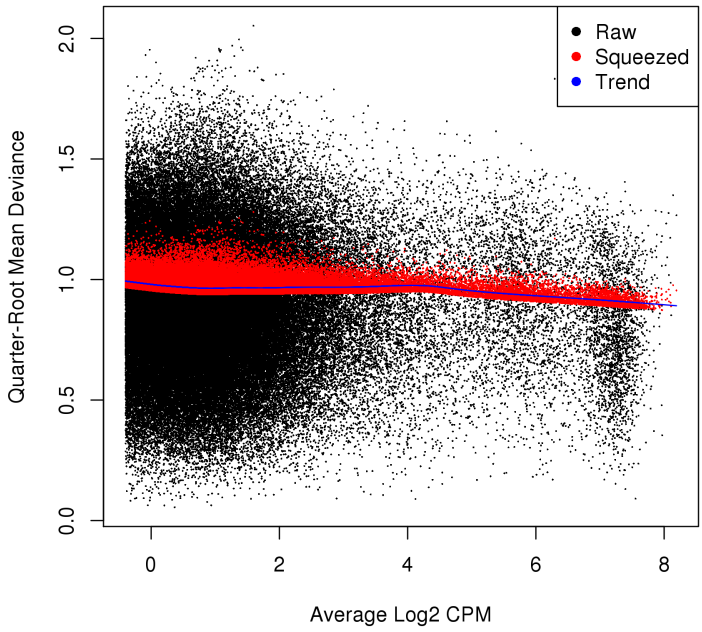
In most cases, the relationship should be monotonic decreasing as the counts become more precise with increasing size. Minor deviations are probably due to the imperfect nature of non-linear normalization. Major increases are indicative of batch effects. For example, a cluster of outliers indicates that 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)
> plotQLDisp(fit)
```

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 the mean/trended value across all bin pairs. This stabilizes the QL dispersion estimates and improves precision for downstream applications.



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$s2.fit$df.prior)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  16.62   17.47   17.47   17.46   17.47   17.47
```

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.frag` objects are required from Chapter 3. Finally, we need to make sure that the csaw package is loaded, e.g., with `library(csaw)`.

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
71499 1.5813245 4.179440 292.2909 3.469359e-13 4.228212e-08
44831 1.3168723 4.679997 256.6783 1.134208e-12 6.911466e-08
65146 1.5586407 3.178065 236.3878 2.390448e-12 9.711036e-08
33017 1.1585524 4.323671 194.0035 1.403760e-11 3.665919e-07
71500 1.1285498 4.442631 192.5033 1.503991e-11 3.665919e-07
2398  1.2286165 4.103014 185.0297 2.136481e-11 4.238809e-07
45686 1.2162879 4.014323 181.2833 2.560017e-11 4.238809e-07
71448 1.2014127 3.856083 179.5807 2.782443e-11 4.238809e-07
30490 1.0858970 4.518359 177.0477 3.153837e-11 4.270751e-07
28128 0.9783828 5.101400 173.5899 3.752016e-11 4.572695e-07
```

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] 5730
```

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

7.2.3 Aggregating small bin pairs

Smaller bins provide greater spatial resolution and can identify sharp differential interactions that would otherwise be lost within larger bin pairs. To demonstrate, a quick-and-dirty analysis of the previously loaded counts for the 100 kbp bin pairs is performed here.

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

Use of smaller bin pairs can be problematic when diffuse interactions are present. Recall that the FDR refers to the proportion of bin pairs that are false positives. However, each bin pair is just an analytical construct. The more relevant error rate is the FDR across the underlying interactions, as this represents actual biology. The FDR over bin pairs can be used as a proxy for the FDR over interactions, if one assumes that each bin pair roughly corresponds to an interaction. This is reasonable for large bin pairs, but is less so when multiple smaller bin pairs are used to represent a diffuse interaction. This can result in misinterpretation of the FDR such that control is lost [Lun and Smyth, 2014].

A typical approach might be to cluster bin pairs in the interaction space to identify the underlying true interaction. One can then combine p-values across all bin pairs in the cluster using Simes' method [Simes, 1986], to obtain a single combined p-value for the cluster. This approach avoids misinterpretation for diffuse interactions as each cluster (and thus interaction) has one p-value. However, this is confounded by the density of the interaction space, particularly at short distances and/or in TADs. The boundaries of each cluster become ambiguous and difficult to interpret, e.g., if a whole TAD is absorbed into a cluster.

Boundary ambiguity can be avoided by performing pre-defined clustering based on large bin pairs. Specifically, a large bin pair is defined and the set of all smaller bin pairs nested therein is defined as a cluster. This is robust to high-density space as the definition of the cluster does not depend on the neighbouring bin pairs. Identification of these clusters can be achieved with the `boxPairs` function, as shown below. Note that the larger bin size must be a multiple of the `width` used for the smaller bins - in this case, 1 Mbp.

```
> matched <- boxPairs(reference=bin.size, smaller=smaller.data, param=mm.param)
```

The p-values for all of the smaller bin pairs can then be combined using Simes' method. The resulting combined p-value represents the evidence against the global null hypothesis for the larger bin pair, i.e., there are no significant differences in any of the smaller bin pairs nested within the larger bin pair. The BH method is applied to the combined p-values to control the FDR across all larger bin pairs/interactions. Rejection of the global null indicates that there is a change and that the larger bin pair is worth further investigation.

```
> tabcom <- combineTests(matched$indices$smaller, result.small$table)
> head(tabcom)
```

```
        logFC     logCPM       PValue          FDR
1 -0.72365050 -1.4307054 3.296941e-02 5.123690e-02
2  0.00495853  1.5307413 2.728477e-02 4.353180e-02
3 -0.07439864 -0.2757410 4.523166e-01 5.009784e-01
4 -0.03679576  1.0005614 1.512522e-08 1.897792e-07
5  0.32862299 -1.3744224 1.244442e-01 1.632958e-01
6 -0.13654721 -0.6613248 1.156879e-05 5.198926e-05
```

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

```
[1] 5460
```

Of course, bin clustering will not be optimal for diffuse interactions. One could obtain larger counts with larger bin pairs, given that spatial resolution is not an issue for diffuse interactions. Nonetheless, bin clustering will protect against misinterpretation when both sharp and diffuse interactions are present. The use of small bin pairs will allow detection of the former, while bin clustering will prevent distortion of the FDR by the latter. This method is also a good introduction to the next section, which might be more useful.

7.2.4 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. Robustness can be provided by combining the differential testing results between small and large bin pairs. All smaller bin pairs that are nested within each of the larger bin pairs are identified using the `boxPairs` function. This yields identifiers for each bin pair where `smaller` bin pairs are nested in `larger` bin pairs with the same ID.

```
> matched <- boxPairs(bin.size, larger=data, smaller=smaller.data, param=mm.param)
> ldex <- matched$indices$larger
> sdex <- matched$indices$smaller
```

Each larger 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 as peviously described. The idea is to provide a compromise between larger counts and spatial resolution in the final combined p-value.

A weighted version of Simes' method is used to compute the combined p-value [Benjamini and Hochberg, 1997]. For each larger bin pair, the weight of the p-value for that bin pair is the same as the combined weight of all the smaller nested bin pairs. This means that the analysis with the larger bin pair has the same contribution to the final combined value as that for the smaller bin pairs. The aim is to avoid increasing the contribution of the latter simply because there are more smaller bin pairs covering the interaction space.

```
> weight <- c(1/counts(matched$pairs)[sdex,"smaller"],
+     1/counts(matched$pairs)[ldex,"larger"])
```

The code snippet above makes sense as the count matrix from the `DIList` in `boxPairs` records the number of nested bin pairs at each bin size. So, as the number of nested bin pairs increases, the weight assigned to each bin pair decreases. Weighted calculation of the combined p-value is then performed using the `combineTests` function in the csaw package.

```
> result.com <- combineTests(ids=c(sdex, ldex),
+     tab=rbind(result.small$table, result$table), weight=weight)
> head(result.com)
```

```
        logFC    logCPM       PValue          FDR
1 -0.72365050 -1.430705 3.296941e-02 2.081569e-01
2  0.13386611  4.685467 3.512986e-04 8.711327e-03
3 -0.11394037  3.066442 7.284887e-02 3.233572e-01
4  0.05831686  4.162275 3.025045e-08 4.801864e-06
5  0.33276441  1.835789 6.037505e-04 1.324918e-02
6 -0.18763932  2.793479 2.313758e-05 9.897199e-04
```

The BH method is then applied to the combined p-values. The FDR refers to the proportion of larger bin pairs that are false discoveries. Like before, misinterpretation of the FDR is mitigated by nesting the smaller bin pairs within their larger counterparts. Note the difference in the results after combining analyses from large and small bin pairs, relative to the analyses using only one bin size. An increase in the number of detections suggests that the different bin sizes complement each other. Sharp changes in the interaction space are detected with small bins, whereas diffuse changes are detected with larger bins.

```
> sum(result.com$FDR <= 0.05)
```

```
[1] 8787
```

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.2 <- anchors(matched$pairs)
> tx.2 <- targets(matched$pairs)
> final.2 <- data.frame(anchor.chr=seqnames(ax.2), anchor.start=start(ax.2),
+     anchor.end=end(ax.2), target.chr=seqnames(tx.2), target.start=start(tx.2),
+     target.end=end(tx.2), result.com)
> o2 <- order(final.2$PValue)
> write.table(final.2[o2,], file="results.2.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

7.2.5 Reporting nested bin pairs

Whenever nesting is involved, it may be convenient to identify the top-ranked (smaller) nested bin pair within each of the (larger) parent bin pairs. Here, the top-ranked bin pair is defined as the one with the smallest individual p-value. This means that any high-resolution changes nested within a parent bin pair can be easily identified. However, users should always keep in mind that the FDR is computed with respect to the parent bin pairs.

```
> inside <- getBestTest(sdex, result.small$table)
> best.vec <- rep(NA, nrow(matched$pairs))
> best.vec[as.integer(rownames(inside))] <- inside$best
> head(best.vec)

[1]   1   4 111 168 318 204

> ax.3 <- as.data.frame(anchors(smaller.data))[best.vec,]
> tx.3 <- as.data.frame(targets(smaller.data))[best.vec,]
> nested <- data.frame(anchor.start=ax.3$start, anchor.end=ax.3$end,
+     target.start=tx.3$start, target.end=tx.3$end,
+     result.small$table[best.vec,c("logFC", "F")])
> head(nested)

    anchor.start anchor.end target.start target.end      logFC         F
1        3004106    3100835            1    3004109 -0.7236505  4.547385
4        3100832    3194461      3100832    3194461  0.4246292 12.463228
111      4401407    4500268      3399155    3501374  0.6343101  7.766902
168      4693997    4801993      4500265    4599273 -0.8542006 40.377818
318      5599281    5695146      3004106    3100835  1.0042754  9.072117
204      5001372    5100850      4899082    5001375 -0.5203174 30.628478

> final.3 <- data.frame(final.2, nest=nested)
> write.table(final.3[o2,], file="results.3.tsv", sep="\t", quote=FALSE, row.names=FALSE)
```

The above code only reports the top-ranked nested bin pair for each parent. 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 `matched` and `data`. Any interesting nested changes can then be interactively identified for a given parent.

7.3 Visualization with plaid plots

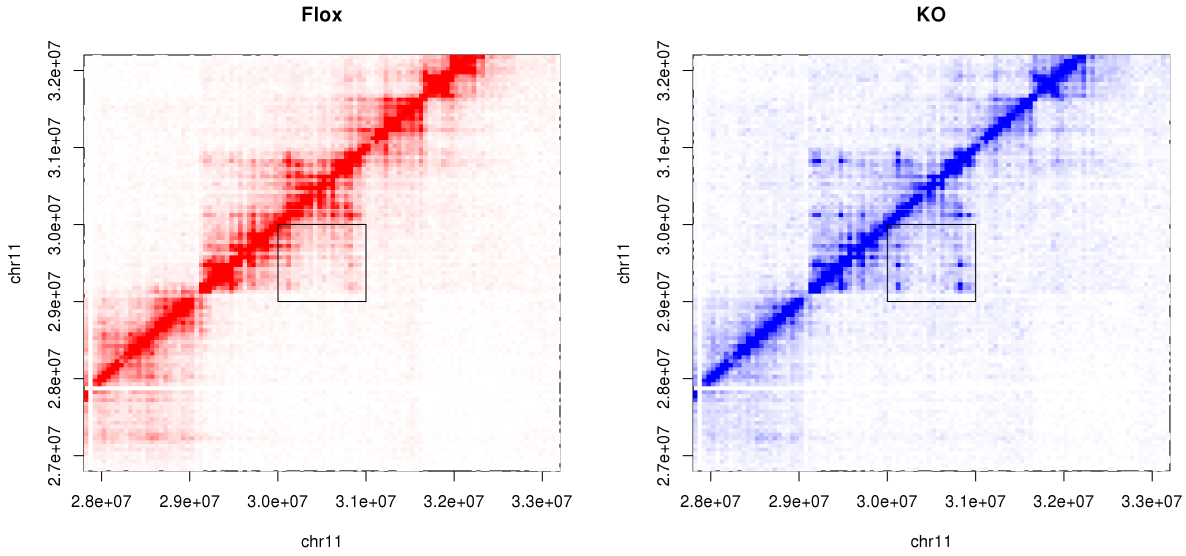
7.3.1 Using conventional plaid plots

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 colour of the pixel is proportional to the number of read pairs mapped between the interacting loci.

```
> chosen <- o2[1]
> expanded.a <- resize(ax.2[chosen], fix="center", width=bin.size*5)
> expanded.t <- resize(tx.2[chosen], fix="center", width=bin.size*5)
> cap1 <- 100
> cap3 <- cap1*totals(data)[3]/totals(data)[1]
> plotPlaid(input[1], anchor=expanded.a, target=expanded.t, cap=cap1,
+     width=5e4, col="red", param=mm.param, main="Flox")
> rect(start(ax.2[chosen]), start(tx.2[chosen]), end(ax.2[chosen]), end(tx.2[chosen]))
```

```
> plotPlaid(input[3], anchor=expanded.a, target=expanded.t, cap=cap3,
+     width=5e4, col="blue", param=mm.param, main="KO")
> rect(start(ax.2[chosen]), start(tx.2[chosen]), end(ax.2[chosen]), end(tx.2[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.



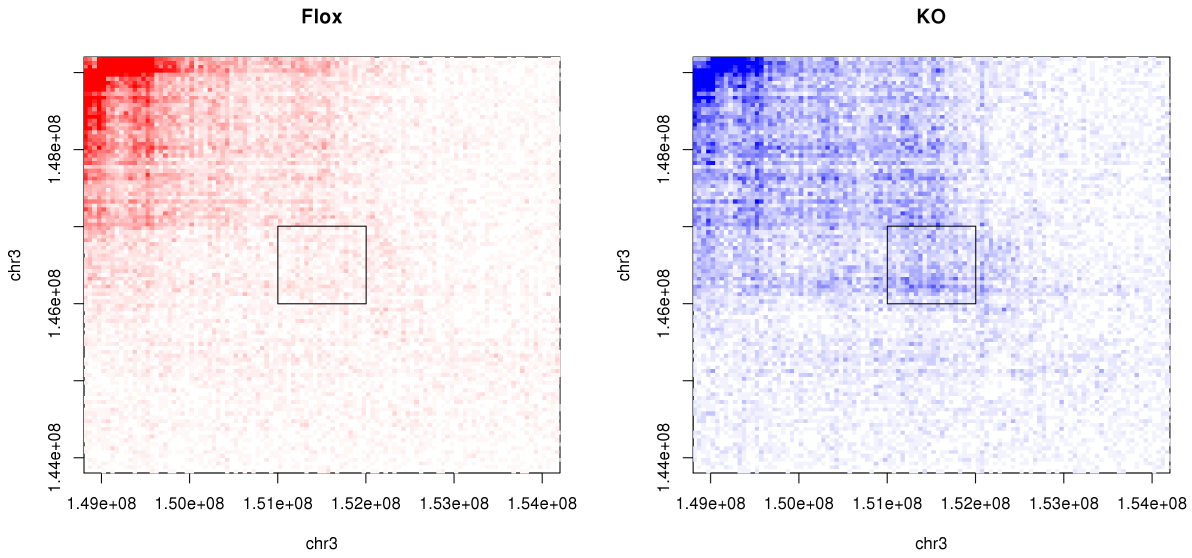
The `cap` value controls the relative scale of the colours. Any pixel with a count above `cap` will be set at the maximum colour intensity. This ensures that a few high-count regions do not dominate the plot. A smaller `cap` is necessary for smaller libraries so that the intensity of the colours is comparable. The actual colour can be controlled 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[1]
> expanded.a <- resize(ax[chosen], fix="center", width=bin.size*5)
```

```
> expanded.t <- resize(tx[chosen], fix="center", width=bin.size*5)
> cap1 <- 30
> cap3 <- cap1*totals(data)[3]/totals(data)[1]
> plotPlaid(input[1], anchor=expanded.a, target=expanded.t, cap=cap1,
+     width=5e4, param=mm.param, col="red", main="Flox")
> rect(start(ax[chosen]), start(tx[chosen]), end(ax[chosen]), end(tx[chosen]))
> plotPlaid(input[3], anchor=expanded.a, target=expanded.t, cap=cap3,
+     width=5e4, col="blue", param=mm.param, main="KO")
> rect(start(ax[chosen]), start(tx[chosen]), end(ax[chosen]), end(tx[chosen]))
```

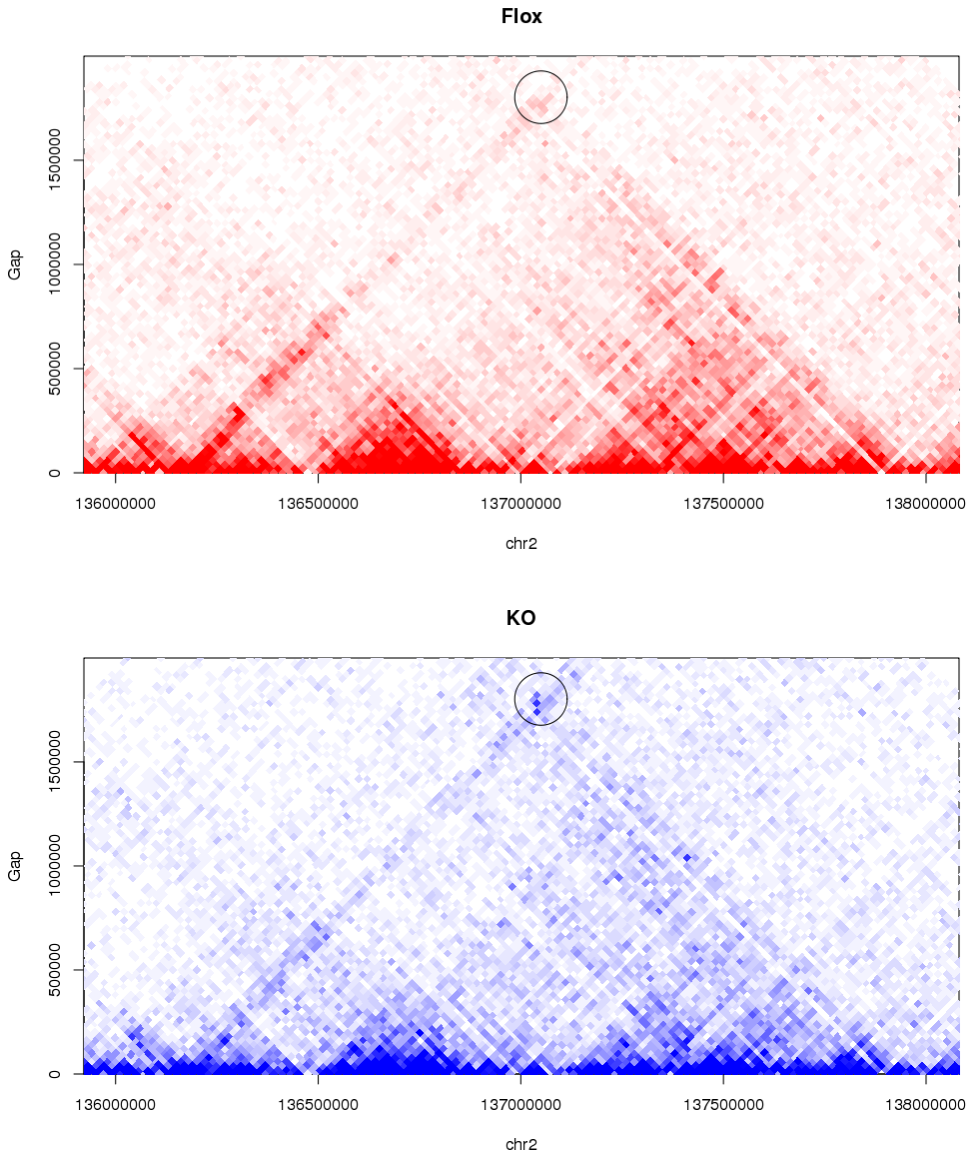


7.3.2 Using rotated plaid plots

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 coloured 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 <- o2[5]
> example <- tx.2[chosen]
> end(example) <- end(ax.2[chosen])
> nest.mid.a <- (ax.3$start[chosen]+ax.3$end[chosen])/2
> nest.mid.t <- (tx.3$start[chosen]+tx.3$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=2e4,
```

```
+     main="Flox", col="Red", cap=cap1)
> points(nest.mid, nest.gap, cex=7)
> rotPlaid(input[3], mm.param, region=example, width=2e4,
+     main="KO", col="blue", cap=cap3)
> points(nest.mid, nest.gap, cex=7)
```

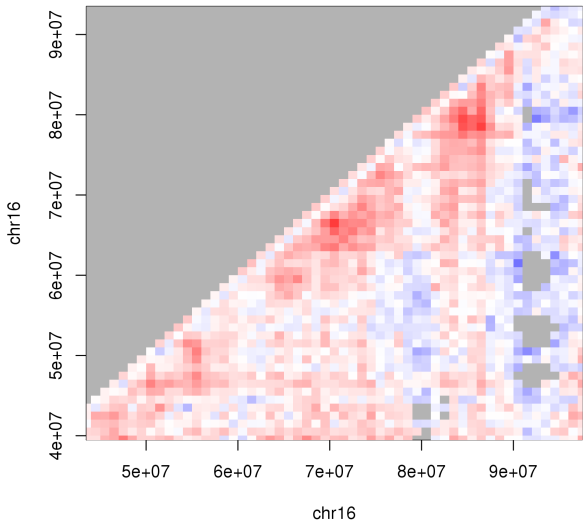


The circle marks the area of the interaction space that corresponds to the top-ranked nested bin pair within the `chosen` larger bin pair. An increase in the interaction intensity is clearly observed in the KO condition. This sharp change would not be observed with larger bin pairs, where the final count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

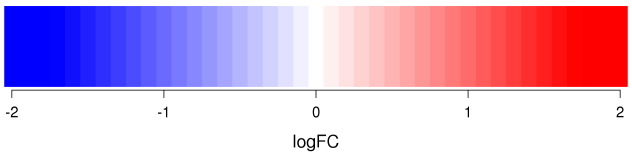
```
> chosen <- o[2]
> expanded.a <- resize(ax[chosen], fix="center", width=5e7)
> expanded.t <- resize(tx[chosen], fix="center", width=5e7)
> colfun <- plotDI(data, result$table$logFC, expanded.a, expanded.t, diag=FALSE)
```



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 colours. This can be used to generate a custom colour bar, as shown below. A similar function that maps counts to colours 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)
```



Chapter 8

Epilogue

8.1 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.

8.2 Session information

```
> sessionInfo()

R Under development (unstable) (2014-12-14 r67167)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] csaw_1.1.16                     org.Mm.eg.db_3.0.0
 [3] RSQLite_1.0.0                   DBI_0.3.1
```

```
 [5] AnnotationDbi_1.29.13              Biobase_2.27.1
 [7] BSgenome.Mmusculus.UCSC.mm10_1.4.0 BSgenome.Hsapiens.UCSC.hg19_1.4.0
 [9] BSgenome_1.35.16                   rtracklayer_1.27.7
[11] Biostrings_2.35.7                  XVector_0.7.3
[13] locfit_1.5-9.1                     edgeR_3.9.9
[15] limma_3.23.8                       diffHic_0.2.3
[17] GenomicRanges_1.19.33              GenomeInfoDb_1.3.12
[19] IRanges_2.1.35                     S4Vectors_0.5.16
[21] BiocGenerics_0.13.4

loaded via a namespace (and not attached):
 [1] base64enc_0.1-2          BatchJobs_1.5            BBmisc_1.8
 [4] BiocParallel_1.1.11      biomaRt_2.23.5           bitops_1.0-6
 [7] brew_1.0-6               checkmate_1.5.1          codetools_0.2-9
[10] digest_0.6.8             fail_1.2                 foreach_1.4.2
[13] GenomicAlignments_1.3.25 GenomicFeatures_1.19.10  grid_3.2.0
[16] iterators_1.0.7          KernSmooth_2.23-13       lattice_0.20-29
[19] RCurl_1.95-4.5           rhdf5_2.11.6             Rsamtools_1.19.26
[22] sendmailR_1.2-1          splines_3.2.0            statmod_1.4.20
[25] stringr_0.6.2            tools_3.2.0              XML_3.98-1.1
[28] zlibbioc_1.13.0
```

8.3 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.

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

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.

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.

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.

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, 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.