Orchestrating Hi-C analysis with Bioconductor

Table of contents

Welcome

Package: OHCA **Authors:** Jacques Serizay [aut, cre] **Compiled:** 2023-11-07 **Package version:** 0.99.0 **R version:** R Under development (unstable) (2023-11-02 r85465) **BioC version:** 3.19 **License:** MIT + file LICENSE

This is the landing page of the **"Orchestrating Hi-C analysis with Bioconductor"** book. **The primary aim of this book is to introduce the `R` user to Hi-C analysis**. This book starts with key concepts important for the analysis of chromatin conformation capture and then presents `Bioconductor` tools that can be leveraged to process, analyze, explore and visualize Hi-C data.

Installation & requirements

General audience

This books aims to demonstrate how to pre-process, parse and investigate Hi-C data in `R`. For this reason, a significant portion of this book consists of executable R code chunks. To be able to reproduce the examples demonstrated in this book and go further in the analysis of *your* ***real*** *datasets*, you will need to rely on several dependencies.

- `R >= 4.3` is required. You can check R version by typing `version` in an R console or in RStudio. If you do not have `R >= 4.3` installed, you will need to update your `R` version, as most extra dependencies will require `R >= 4.3`.

> **i** Installing R 4.3
>
> Detailed instructions are available here to install `R 4.3` on a Linux machine (Ubuntu 22.04).
> Briefly, to install pre-compiled version of `R 4.3.0`:

```
# This is adapted from Posit (https://docs.posit.co/resources/install-r/)
export R_VERSION=4.3.0

# Install curl and gdebi-core
sudo apt update -qq
sudo apt install curl gdebi-core -y

# Fetching the `.deb` install file from Posit repository
curl -O https://cdn.rstudio.com/r/ubuntu-2204/pkgs/r-${R_VERSION}_1_amd64.deb

# Install R
sudo gdebi r-${R_VERSION}_1_amd64.deb --non-interactive -q

# Optional: create a symlink to add R to your PATH
sudo ln -s /opt/R/${R_VERSION}/bin/R /usr/local/bin/R
```

If you have some issues when installing the Hi-C packages listed below, you may need to install the following system libraries:

```
sudo apt update -qq
sudo apt install -y \
    automake make cmake fort77 gfortran \
    bzip2 unzip ftp build-essential \
    libc6 libreadline-dev \
    libpng-dev libjpeg-dev libtiff-dev \
    libx11-dev libxt-dev x11-common \
    libharfbuzz-dev libfribidi-dev \
    libfreetype6-dev libfontconfig1-dev \
    libbz2-dev liblzma-dev libtool \
    libxml2 libxml2-dev \
    libzstd-dev zlib1g-dev \
    libdb-dev libglu1-mesa-dev \
    libncurses5-dev libghc-zlib-dev libncurses-dev \
    libpcre3-dev libxml2-dev libblas-dev libzmq3-dev \
    libssl-dev libcurl4-openssl-dev \
    libgsl-dev libeigen3-dev libboost-all-dev \
    libgtk2.0-dev xvfb xauth xfonts-base apt-transport-https \
    libhdf5-dev libudunits2-dev libgdal-dev libgeos-dev \
    libproj-dev libnode-dev libmagick++-dev
```

- `Bioconductor >= 3.18` is also required. You can check whether `Bioconductor` is available and its version in `R` by typing `BiocManager::version()`. If you do not have `BiocManager` >= 3.18 installed, you will need to update it as follows:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.18")
```

- You will also need important packages, which will be described in length in this book. The following `R` code will set up most of the extra dependencies:

```
BiocManager::install("HiCExperiment", ask = FALSE)
BiocManager::install("HiCool", ask = FALSE)
BiocManager::install("HiContacts", ask = FALSE)
```

```
BiocManager::install("HiContactsData", ask = FALSE)
BiocManager::install("fourDNData", ask = FALSE)
BiocManager::install("DNAZooData", ask = FALSE)
```

Developers

For developers or advanced R users, the `devel` versions of these packages can be installed along with their dependencies:

```
install.packages("pak", repos = "https://r-lib.github.io/p/pak/devel/")
pak::pkg_install("js2264/HiCExperiment", ask = FALSE, dependencies = c("Depends", "Imports",
pak::pkg_install("js2264/HiCool", ask = FALSE, dependencies = c("Depends", "Imports", "Sugge
pak::pkg_install("js2264/HiContacts", ask = FALSE, dependencies = c("Depends", "Imports", "S
pak::pkg_install("js2264/HiContactsData", ask = FALSE, dependencies = c("Depends", "Imports"
pak::pkg_install("js2264/fourDNData", ask = FALSE, dependencies = c("Depends", "Imports", "S
pak::pkg_install("js2264/DNAZooData", ask = FALSE, dependencies = c("Depends", "Imports", "S
```

Docker image

If you have `docker` installed, the easiest approach would be to run the following command in a `shell` terminal:

```
docker run -it ghcr.io/js2264/ohca:latest R
```

This will fetch a `docker` image with the latest development versions of the aforementioned packages pre-installed, and initiate an interactive R session.

Reproducibility

Building book

The OHCA book has been rendered in R thanks to a number of packages, including but not only:

- `BiocBook`
- `devtools`
- `quarto`
- `rebook`

To build this book locally, you can run:

Listing 0.1 `bash`

```
git clone git@github.com:js2264/OHCA.git && cd OHCA
quarto render
```

> **Warning**
>
> All dependencies listed above will be required!

The actual rendering of this book is done by GitHub Actions, and the rendered static website is hosted by GitHub Pages.

Session info

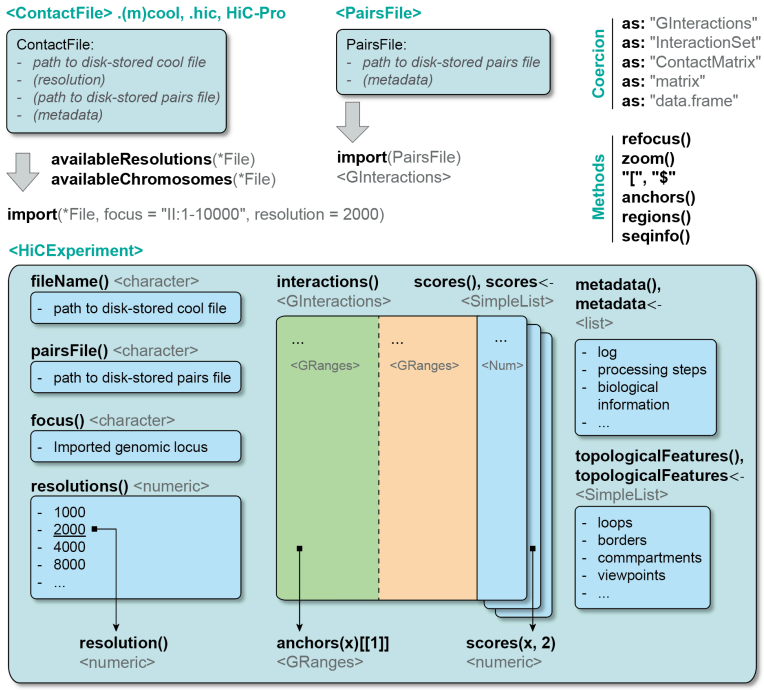
Click to expand

Preamble

Hi-C is an experimental method to quantify spatial interactions between any pair of genomic loci. While a number of command-line interfaces (CLI) exist to process and manipulate Hi-C data (e.g. `cooler` (@Abdennur_2019), `juicer` (@Durand_2016) and HiC-Pro (@Servant_2015)), they generally suffer from several limitations often found in emerging genomics techniques:

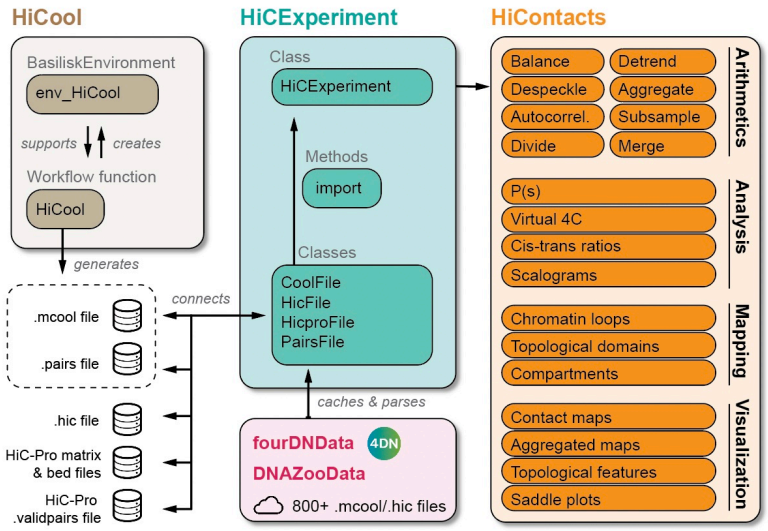
- **No genomic representation of Hi-C processed data**: the existing CLIs can efficiently parse Hi-C data as a numerical matrix and perform a few standard quantitative operations (e.g. contact matrix binning and normalization, dimensionality reduction, etc). However, they systematically fail to **represent a Hi-C contact matrix as a genomic object**. Qualitative analyses (e.g. intersecting chromatin loops with genomic features, finding genes overlapping with domains, etc) therefore remain extremely tedious.
- **No format-agnostic analysis libraries**. Three competing file format standards (`.(m)cool`, `.hic` and `HiC-Pro` files) currently exist to store Hi-C processed data and dedicated CLIs propose sets of tools specifically working with their corresponding Hi-C processed data file format. This has curbed the development of generic Hi-C data analysis libraries by favoring the emergence of several redundant tools.
- **Lack of integration within a biology-oriented community**. While rapid development of Hi-C analysis methodology is ongoing, it is primarily driven by small-scale teams rather than by a community as a whole. This oriented development is less likely to fulfill the needs met by other investigators.

In this book, we provide an **overview of a set of tools that enable processing, visualization and in-depth investigation of Hi-C data in R, ensuring intuitive integration of Hi-C data in the existing Bioconductor ecosystem**. We introduce a high-level `HiCExperiment` data structure to represent Hi-C data, directly extending robust, pre-existing core genomic classes offered by `Bioconductor`. This guarantees a stable and intuitive Hi-C data representation in R as a genomic entity, which is highly interoperable and can be used by all existing analysis packages in R.



On top of the `HiCExperiment` data structure, the `HiContacts` package offers extended functionalities to perform matrix-centric and interaction-centric analysis directly on `HiCExperiment` objects and provides powerful visualization tools specifically designed for Hi-C data to facilitate exploratory data analysis. In addition, the `HiCool` package implements a processing workflow based on a lightweight library to process raw Hi-C data into binned Hi-C contact matrices ready to be imported as `HiCExperiment` objects. Finally, the `fourDNData` and `DNAZooData` packages offer a gateway to major public data

repositories directly in R.



References

Part I

Fundamentals concepts

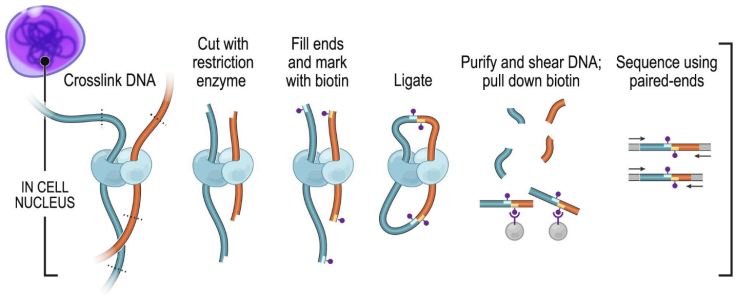
1 Hi-C pre-processing steps

> **Aims**
>
> This chapter introduces the reader to general Hi-C experimental and computational steps to perform the pre-processing of Hi-C. This encompasses read alignment, pairs generation and filtering and pairs binning into a contact matrix file.

1.1 Experimental considerations

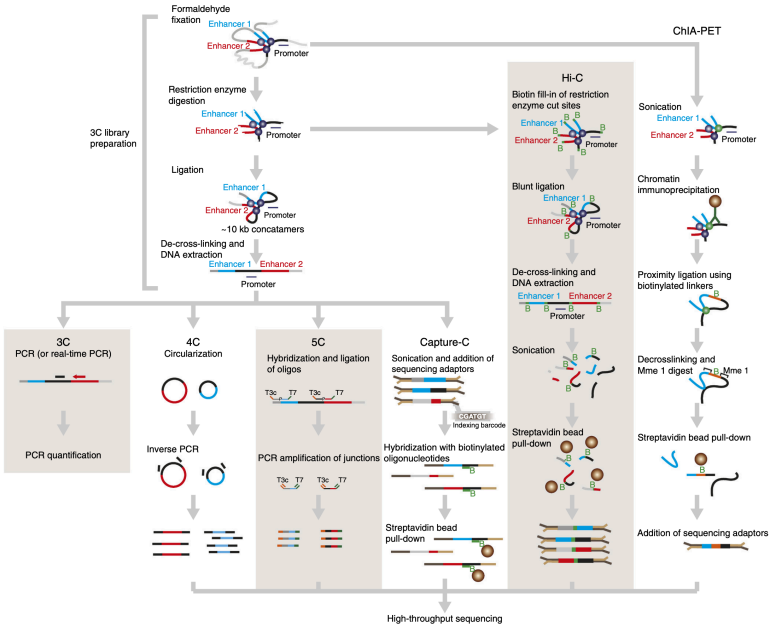
1.1.1 Experimental approach

The Hi-C procedure (@Lieberman_Aiden_2009) stems from the clever combination of high-throughput sequencing and Chromatin Conformation Capture (3C) experimental approach (@Dekker_2002).
In Hi-C, chromatin is crosslinked within intact nuclei and enzymatically digested (usually with one or several restriction enzymes, but Hi-C variants using MNase or DNase exist). End-repair introduces biotinylated dNTPs and is followed by religation, which generates chimeric DNA fragments consisting of genomic loci originally lying in spatial proximity, usually crosslinked to a shared protein complex. After religation, DNA fragments are sheared, biotin-containing fragments are pulled-down and converted into a sequencing library.



1.1.2 C variants

A number of C variants have been proposed since the publication of the original 3C method (reviewed by @Davies_2017), the main ones being Capture-C and ChIA-PET (see procedure below).



Capture-C is useful to quantify interactions between a set of regulatory elements of interest. ChIA-PET, on the other hand, can identify interactions mediated by a specific protein of interest. Finally, an increasing number of Hi-C approaches rely on long-read sequencing (e.g. @Deshpande_2022, @Tavares_Cadete_2020) to identify clusters of 3D contacts.

1.1.3 Sequencing

Hi-C libraries are traditionally sequenced with short-read technology, and are by essence paired-end libraries. For this reason, the end result of the experimental side of the Hi-C consists of **two fastq files**, each one containing sequences for one extremity of the DNA fragments purified during Hi-C. These are the two files we need to move on to the computational side of Hi-C.

Fastq files are plain text files (usually compressed, with the `.gz` extension). They are generated by the sequencing machine during a sequencing run, and for Hi-C, necessarily come in pairs, generally called `*_R1.fq.gz` and `*_R2.fq.gz`.

Here is the first read listed in `sample_R1.fq.gz` file:

Listing 1.1 `sample-R1.fq.gz`

```
@SRR5399542.1.1 DH1DQQN1:393:H9GEWADXX:1:1101:1187:2211 length=24
CAACTTCAATACCAGCAGCAGCAA
+
CCCFFFFFHHHHHJJJJJIJJJJJ
```

And here is the first read listed in `sample_R2.fq.gz` file:

Listing 1.2 `sample-R2.fq.gz`

```
@SRR5399542.1.1 DH1DQQN1:393:H9GEWADXX:1:1101:1187:2211 length=24
GCTGTTGTTGTTGTTGTATTTGCA
+
@@@FFFFFFHHHHIJJIJJHIIEH
```

These two reads are the first listed in their respective file. Notice how they bear the same name (first line): **they form a pair**. The second line corresponds to the sequence read by the sequencer, the third line is a single `+` separator, and the last line indicates the per-base sequencing quality following a nebulous cypher.

1.2 Hi-C file formats

Two important output files are typically generated during Hi-C data pre-processing:

- A **"pairs" file**;
- A **binned "contact matrix"** file

We will now describe the structure of these different types of files. Directly jump to the next chapter if you want to know more about importing data from a contact matrix or a pairs file in R.

1.2.1 Pairs files

A "pairs" file (optionally, but generally filtered and sorted) is the direct output of processing Hi-C fastq files. It stores information about putative proximity contacts identified by digestion/religation, in the lossless, human-readable, indexable format: the `.pairs` format.

A `.pairs` file is organized in a `header` followed by a `body`:

- `header`: starts with `#`
  - Required entries
    - First line: `## pairs format v1.0`
    - `#columns`: column contents and ordering (e.g. `#columns: readID chr1 pos1 chr2 pos2 strand1 strand2 <column_name> <column_name> ...`)
    - `#chromsize`: chromosome names and their size in bp, one chromosome per line, in the same order that defines ordering between mates (e.g. `#chromsize: chr1 230218`). Chromosome order is actually defined by this header, not by the order of pairs listed in the `body`!
  - Optional entries with reserved header keys (`sorted`, `shape`, `command`, `genome_assembly`)

    * `#sorted`: to indicate the sorting mechanism (e.g. `#sorted: chr1-chr2-pos1-pos2`, `#sorted: chr1-pos1`, `#sorted: none`)
    * `#shape`: to specify whether the matrix is stored as upper triangle or lower triangle (`#shape: upper triangle`, `#shape: lower triangle`)
    * `#command`: to specify any command, e.g. the command used to generate the pairs file (`#command: bam2pairs mysample.bam mysample`)
    * `#genome_assembly`: to specify the genome assembly (e.g. `#genome_assembly: hg38`)

- `body`: tab-separated columns
  - 7 reserved (4 of them required) columns:
    * `readID, chr1, pos1, chr2, pos2, strand1, strand2`
    * Columns 2-5 (`chr1, pos1, chr2, pos2`) are required and cannot have missing values
    * For column 1, 6 & 7: missing values are annotated with a single-character dummy (`.`)
  - 2 extra reserved, optional column **names**:
    * `frag1`, `frag2`: restriction enzyme fragment index used by `Juicer`
  - Any number of optional columns can be added

More information about the conventions related to this text file are provided by the 4DN consortium, which originally formalized the specifications of this file format.

1.2.2 Binned contact matrix files

1.2.2.1 Binning pairs into a matrix

The action of "binning" a `.pairs` file into a contact matrix consists in (1) *discretizing* a genome reference into genomic *bins*, (2) attributing bins for each pair's extremity and (3) computing the interaction frequency between any pair of genomic bins, i.e. the "contact matrix".

Listing 1.3 `sample.pairs`

```
## pairs format v1.0
#sorted: chr1-chr2-pos1-pos2
#shape: upper triangle
#genome_assembly: hg38
#chromsize: chr1 249250621
#chromsize: chr2 243199373
#chromsize: chr3 198022430
...
#columns: readID chr1 pos1 chr2 pos2 strand1 strand2
EAS139:136:FC706VJ:2:2104:23462:197393 chr1 10000 chr1 20000 + +
EAS139:136:FC706VJ:2:8762:23765:128766 chr1 50000 chr1 70000 + +
EAS139:136:FC706VJ:2:2342:15343:9863 chr1 60000 chr2 10000 + +
EAS139:136:FC706VJ:2:1286:25:275154 chr1 30000 chr3 40000 + -
```

For instance, here is a dummy `.pairs` file with a total of 5 pairs:

Listing 1.4 `dummy.pairs`

```
## pairs format v1.0
#sorted: chr1-chr2-pos1-pos2
#columns: readID chr1 pos1 chr2 pos2 strand1 strand2
#chromsize: chr1 389
. chr1 162 chr1 172 . .
. chr1 180 chr1 192 . .
. chr1 183 chr1 254 . .
. chr1 221 chr1 273 . .
. chr1 254 chr1 298 . .
```

Note that this genome reference is made of a single chromosome (`chr1`), very short (length of `389`). By binning this chromosome in 100bp-wide bins (`100` bp is the *`resolution`*), one would obtain the following four `bins`:

Each pair extremity can be changed to an integer indicating the

Listing 1.5 `bins.bed`

```
<chr>  <pos> <bin>

chr1   1     100

chr1   101   200

chr1   201   300

chr1   301   389
```

position of the `bin` it falls in, e.g. for the left-hand extremity of the pairs file printed hereinabove (`bin1`):

```
<chr1>  <pos1>  ->  <bin1>
chr1    162     ->  2
chr1    180     ->  2
chr1    183     ->  2
chr1    221     ->  3
chr1    254     ->  3
```

Similarly for the right-hand extremity of the pairs file (`bin2`):

```
<chr2>  <pos2>  ->  <bin2>
chr1    172     ->  chr1 2
chr1    192     ->  chr1 2
chr1    254     ->  chr1 3
chr1    273     ->  chr1 3
chr1    298     ->  chr1 3
```

By pasting side-to-side the left-hand and right-hand extremities of each pair, the `.pairs` file can be turned into something like:

```
<bin1> <bin2>
2      2
2      2
```

```
2       3
3       3
3       3
```

And if we now count the number of each `<bin1> <bin2>` combinaison, adding a third `<count>` column, we end up with a `count.matrix` text file:

Listing 1.6 `count.matrix`

```
<bin1> <bin2>  <count>
2      2       2
2      3       1
3      3       2
```

This `count.matrix` file lists a total of 5 pairs, and in which bin each extremity of each pair is contained. Thus, a count matrix is a lossy file format, as it "rounds up" the position of each pair's extremity to the genomic bin containing it.

This "i-j-x" 3-column format, in which `i-j` relate to a pair of "coordinates" indices (or a pair of genomic bin indices) in a matrix, and `x` relates to a score associated with the pair of indices, is generally called a "COO sparse matrix".

In this context, the `regions.bed` acts as a secondary "dictionary" describing the nature of `i` and `j` indices, i.e. the location of genomic bins.

1.2.2.2 Plain-text matrices: HiC-Pro style

The HiC-Pro pipeline (@Servant_2015) outputs 2 text files: a `regions.bed` file and a `count.matrix` file. They are generated by the exact process explained above.

Together, these two files can describe the interaction frequency between any pair of genomic loci. They are non-binarized text files, and as such are technically human-readable. However, it is relatively hard to get a grasp of these files compared to a plain `.pairs` file, as information regarding genomic bins and

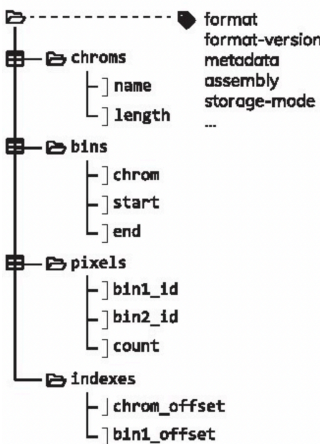
interaction frequencies are stored in separate files. Moreover, because they are non-binarized, these files often end up using a large disk space and cannot be easily indexed. This prevents easy subsetting of the data stored in these files.

`.(m)cool` and `.hic` file formats are two standards addressing these limitations.

1.2.2.3 `.(m)cool` matrices

The `.cool` format has been formally defined in @Abdennur_2019 and is a particular type of `HDF5` (`Hierarchical Data Format`) file. It is an indexed archive file storing rectangular tables called:

- `bins`: containing the same information than the `regions.bed` file;
- `pixels`: containing the same information than the `count.matrix` (each "`pixel`" is a pair of 2 `bins` and has one or several associated `score`s);
- `chroms`: summarizing the order and length of the chromosomes present in a Hi-C contact matrix;
- `indexes`: allowing `random access`, i.e. parsing of only a subset of the data without having to read through the entire set of data.



A single `.pairs` file binned at different resolutions can also be saved into a single, multi-resolution `.mcool` file. `.mcool` essentially consists of nested `.cool` files.

Importantly, as an `HDF5`-based format, `.cool` files are binarized, indexed and highly-compressed. This has two major benefits:

1. **Smaller disk storage footprint**
2. **Rapid subsetting of the data** through random access

Moreover, parsing `.cool` files is possible using `HDF` standard APIs.

1.2.2.4 `.hic` matrices

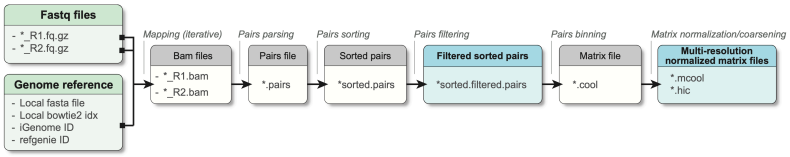
The `.hic` format is another type of binarized, indexed and highly-compressed file (@Durand_2016). It can store virtually the same information than a `.cool` file. However, parsing `.hic` files is not as straightforward as `.cool` files, as it does not rely on a generic file standard. Still, the `straw` library has been implemented in several computing languages to facilitate parsing of `.hic` files (@Durand_2016).

1.3 Pre-processing Hi-C data

1.3.1 Processing workflow

Fundamentally, the main steps performed to pre-process Hi-C are:

1. Separate read mapping
2. Pairs parsing
3. Pairs sorting
4. Pairs filtering
5. Pairs binning into a contact matrix
6. Normalization of contact matrix and multi-resolution matrix generation



In practice, a minimal workflow to pre-process Hi-C data is the following (adapted from @Open2C_2023):

```
## Note these fields have to be replaced by appropriate variables:
##    <index>
##    <input.R1.fq.gz>
##    <input.R2.fq.gz>
##    <chromsizes.txt>
##    <prefix>
bwa mem2 -SP5M <index> <input.R1.fq.gz> <input.R2.fq.gz> \
    | pairtools parse -c <chromsizes.txt> \
    | pairtools sort \
    | pairtools dedup \
    | cooler cload pairs -c1 2 -p1 3 -c2 4 -p2 5 <chromsizes.txt>:10000 - <prefix>.cool
cooler zoomify --balance --nproc 32 --resolutions 5000N --out <prefix>.mcool <prefix>.cool
```

Several pipelines have been developed to facilitate Hi-C data pre-processing. A few of them stand out from the crowd:

- `nf-distiller`: a combination of an aligner + `pairtools` + `cooler`
- `HiC-pro` (@Servant_2015)
- `Juicer` (@Durand_2016)

> **Note**
>
> For larger genomes (> 1Gb) with more than few tens of M of reads per fastq (e.g. > 100M), we recommend pre-processing data on an HPC cluster. Aligners, pairs processing and matrix binning can greatly benefit from parallelization over multiple CPUs (@Open2C_2023)).
> To scale **up** data pre-processing, we recommend to rely on an efficient read mapper such as `bwa`, followed by pairs parsing, sorting and deduplication with `pairtools` and binning with `cooler`.

1.3.2 hicstuff: lightweight Hi-C pipeline

`hicstuff` is an integrated workflow to process Hi-C data. Some advantages compared to solutions mentioned above are its simplicity, flexibility and lightweight. For shallow sequencing or Hi-C on smaller genomes, it efficiently parses fastq reads and processes data into binned contact matrices with a single terminal command.

`hicstuff` provides both a command-line interface (`CLI`) and a python `API` to process fastq reads into a binned contact matrix. A processing pipeline can be launched using the standard command `pipeline` as follows:

```
## Note these fields have to be replaced by appropriate variables:
##    <hicstuff-options>
##    <genome.fa>
##    <input.R1.fq.gz>
##    <input.R2.fq.gz>
hicstuff pipeline \
   <hicstuff-options> \
   --genome <genome.fa> \
   <input.R1.fq.gz> \
   <input.R2.fq.gz>
```

`hicstuff` documentation website is available here: https://hicstuff.readthedocs.io/ to read more about available options and internal processing steps.

1.3.3 HiCool: hicstuff within R

`hicstuff` is available as a standalone (`conda install -c bioconda hicstuff` it!). It is also shipped in an R package: `HiCool`. Thus, `HiCool` can process fastq files directly within an R console.

1.3.3.1 Executing HiCool

To demonstrate this, we first fetch example `.fastq` files:

```
library(HiContactsData)
r1 <- HiContactsData(sample = 'yeast_wt', format = 'fastq_R1')
r2 <- HiContactsData(sample = 'yeast_wt', format = 'fastq_R2')
```

We then load the `HiCool` library and execute the main `HiCool` function.

```
library(HiCool)
HiCool(
    r1,
    r2,
    restriction = 'DpnII,HinfI',
    resolutions = c(4000, 8000, 16000),
    genome = 'R64-1-1',
    output = './HiCool/'
)
```

1.3.3.2 HiCool arguments

Several arguments can be passed to `HiCool` and some are worth mentioning them:

- `restriction`: (default: `"DpnII,HinfI"`)
- `resolutions`: (default: `NULL`, automatically inferring resolutions based on genome size)
- `iterative`: (default: `TRUE`)
- `filter`: (default: `TRUE`)
- `balancing_args`: (default: `" --cis-only --min-nnz 3 --mad-max 7 "`)
- `threads`: (default: `1L`)

Other `HiCool` arguments can be listed by checking `HiCool` documentation in R: `?HiCool::HiCool`.

1.3.3.3 HiCool outputs

We can check the generated output files placed in the `HiCool/` directory.

```
fs::dir_tree('HiCool/')
```

- The `*.pairs` and `*.mcool` files are the pairs and contact matrix files, respectively. **These are the output files the end-user is generally looking for.**
- The `*.html` file is a report summarizing pairs numbers, filtering, etc…
- The `*.log` file contains all output and error messages, as well as the full list of commands that have been executed to pre-process the input dataset.
- The `*.pdf` graphic files provide a visual representation of the distribution of informative/non-informative pairs.

> Tip
>
> All the files generated by a single `HiCool` pipeline execution contain the same 6-letter unique hash to make sure they are not overwritten if re-executing the same command.

1.4 Exploratory data analysis of processed Hi-C files

Once Hi-C raw data has been transformed into a set of processed files, exploratory data analysis is typically conducted following two main routes:

- Data visualization;
- Data investigation.

During the last decade, a number of softwares have been developed to unlock **Hi-C data visualization and investigation**. Here we provide a non-exhaustive list of notable tools developed

throughout the recent years for downstream Hi-C analysis, selected from this longer list.

- 2012-2015:
  - HiTC (2012)
  - HiCCUPS (2014)
  - HiCseg (2014)
  - Fit-Hi-C (2014)
  - HiC-Pro (2015)
  - diffHic (2015)
  - cooltools (2015)
  - HiCUP (2015)
  - HiCPlotter (2015)
  - HiFive (2015)
- 2016-2019:
  - CHiCAGO (2016)
  - TADbit (2017)
  - HiCRep (2017)
  - HiC-DC (2017)
  - GoTHIC (2017)
  - HiCExplorer (2018)
  - Boost-HiC (2018)
  - HiCcompare (2018)
  - HiPiler (2018)
  - coolpuppy (2019)
- 2020-present:
  - Serpentine (2020)
  - CHESS (2020)
  - DeepHiC (2020)
  - Chromosight (2020)
  - Mustache (2020)
  - TADcompare (2020)
  - POSSUM (2021)
  - Calder (2021)
  - HICDCPlus (2021)
  - plotgardener (2021)
  - GENOVA (2021)

All references as well as many other softwares and references are available here.

References

2 Hi-C data structures in R

Aims

This chapter introduces the four main classes offered by `Bioconductor` leveraged to perform Hi-C analysis, describes their structure and how to interact with them:

- `GRanges` (jump to the section)
- `GInteractions` (jump to the section)
- `ContactFile` (jump to the section)
- `HiCExperiment` (jump to the section)

TL;DR

Directly jump to the last section of this chapter to get a visual representation of these data structures.

2.1 `GRanges` class

`GRanges` is a shorthand for `GenomicRanges`, a core class in `Bioconductor`. This class is primarily used to describe genomic ranges of any nature, e.g. sets of promoters, SNPs, chromatin loop anchors, ….
The data structure has been published in the seminal 2015 publication by the `Bioconductor` team (@Huber_2015).

2.1.1 `GRanges` fundamentals

The easiest way to generate a `GRanges` object is to coerce it from a vector of genomic coordinates in the UCSC format (e.g. `"chr2:2004-4853"`):

```
library(GenomicRanges)
gr <- GRanges(c(
    "chr2:2004-7853:+",
    "chr4:4482-9873:-",
    "chr5:1943-4203:+",
    "chr5:4103-5004:+"
))
gr
##  GRanges object with 4 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr2 2004-7853      +
##    [2]     chr4 4482-9873      -
##    [3]     chr5 1943-4203      +
##    [4]     chr5 4103-5004      +
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

A single `GRanges` object can contain one or several **"ranges"**, or genomic intervals. To navigate between these ranges, `GRanges` can be subset using the standard `R` single bracket notation `[`:

```
gr[1]
##  GRanges object with 1 range and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr2 2004-7853      +
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

gr[1:3]
##  GRanges object with 3 ranges and 0 metadata columns:
##        seqnames    ranges strand
```

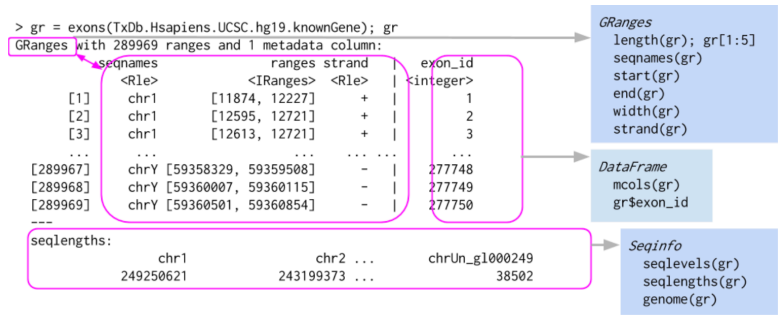
```
##           <Rle> <IRanges>  <Rle>
##    [1]     chr2 2004-7853      +
##    [2]     chr4 4482-9873      -
##    [3]     chr5 1943-4203      +
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

`GenomicRanges` objects aim to provide a natural description of genomic intervals (ranges) and are incredibly versatile. They extend the `data.frame` object and have four required pieces of information:

- `seqnames` (i.e. chromosome names) (accessible with `seqnames()`)
- `start` (accessible with `start()`)
- `end` (accessible with `end()`)
- `strand` (accessible with `strand()`)

```
seqnames(gr)
##  factor-Rle of length 4 with 3 runs
##    Lengths:    1    1    2
##    Values : chr2 chr4 chr5
##  Levels(3): chr2 chr4 chr5

start(gr)
##  [1] 2004 4482 1943 4103

end(gr)
##  [1] 7853 9873 4203 5004

strand(gr)
##  factor-Rle of length 4 with 3 runs
##    Lengths: 1 1 2
##    Values : + - +
##  Levels(3): + - *
```

Here is a graphical representation of a `GRanges` object, taken from Bioconductor course material:



We will now delve into the detailed structure and operability of `GRanges` objects.

2.1.2 `GRanges` metadata

An important aspect of `GRanges` objects is that each entry (range) can have extra optional metadata. This metadata is stored in a rectangular `DataFrame`. Each column can contain a different type of information, e.g. a `numerical` vector, a `factor`, a list, ...

One can directly access this `DataFrame` using the `mcols()` function, and individual columns of metadata using the `$` notation:

```
mcols(gr)
##  DataFrame with 4 rows and 0 columns
mcols(gr)$GC <- c(0.45, 0.43, 0.44, 0.42)
mcols(gr)$annotation <- factor(c(NA, 'promoter', 'enhancer', 'centromere'))
mcols(gr)$extended.info <- c(
    list(c(NA)),
    list(c(date = 2023, source = 'manual')),
    list(c(date = 2021, source = 'manual')),
    list(c(date = 2019, source = 'homology'))
)
mcols(gr)
##  DataFrame with 4 rows and 3 columns
##            GC annotation extended.info
##     <numeric>   <factor>        <list>
##  1       0.45 NA                   NA
```

```
##  2      0.43 promoter     2023,manual
##  3      0.44 enhancer     2021,manual
##  4      0.42 centromere 2019,homology
```

When metadata columns are defined for a `GRanges` object, they are pasted next to the minimal 4 required `GRanges` fields, separated by a `|` character.

```
gr
## GRanges object with 4 ranges and 3 metadata columns:
##       seqnames    ranges strand |        GC annotation extended.info
##          <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

2.1.3 Genomic arithmetics on individual `GRanges` objects

A `GRanges` object primarily describes a set of genomic ranges (*it is in the name!*). Useful genomic-oriented methods have been implemented to investigate individual `GRanges` object from a genomic perspective.

2.1.3.1 Intra-range methods

Standard genomic arithmetics are possible with `GRanges`, e.g. shifting ranges, resizing, trimming, … These methods are referred to as "intra-range" methods as they work *"one-region-at-a-time"*.

> **Note**
>
> - Each range of the input `GRanges` object is modified independently from the other ranges in the following

code chunks.

- Intra-range operations are **endomorphisms**: they all take `GRanges` inputs and always return `GRanges` objects.

- Shifting each genomic range in a `GRanges` object by a certain number of bases:

```
gr
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Shift all genomic ranges towards the "right" (downstream in `+` strand), by 1000bp:
shift(gr, 1000)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames     ranges strand |        GC annotation extended.info
##           <Rle>  <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2  3004-8853      + |      0.45 NA                  <NA>
##    [2]     chr4 5482-10873      - |      0.43 promoter     2023,manual
##    [3]     chr5  2943-5203      + |      0.44 enhancer     2021,manual
##    [4]     chr5  5103-6004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Shift all genomic ranges towards the "left" (upstream in `+` strand), by 1000bp:
shift(gr, -1000)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 1004-6853      + |      0.45 NA                  <NA>
##    [2]     chr4 3482-8873      - |      0.43 promoter     2023,manual
##    [3]     chr5  943-3203      + |      0.44 enhancer     2021,manual
```

```
##    [4]     chr5 3103-4004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

- Narrowing each genomic range in a `GRanges` object by a certain number of bases:

```
gr
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Extract 21st-40th subrange for each range in `gr`:
narrow(gr, start = 21, end = 40)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2024-2043      + |      0.45 NA                  <NA>
##    [2]     chr4 4502-4521      - |      0.43 promoter     2023,manual
##    [3]     chr5 1963-1982      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4123-4142      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

width(narrow(gr, start = 21, end = 40))
##  [1] 20 20 20 20
```

- Resizing each genomic range in a `GRanges` object to a certain number of bases:

```
gr
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
```

```
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Resize `gr` entries to 100, fixed at the start of each range:
resize(gr, 100, fix = "start")
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-2103      + |      0.45 NA                  <NA>
##    [2]     chr4 9774-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-2042      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-4202      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Resize `gr` entries to 100, fixed at the start of each range, disregarding strand in
resize(gr, 100, fix = "start", ignore.strand = TRUE)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-2103      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-4581      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-2042      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-4202      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Resize `gr` entries to 1 bp, fixed at the center of each range:
resize(gr, 1, fix = "center")
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2      4928      + |      0.45 NA                  <NA>
##    [2]     chr4      7177      - |      0.43 promoter     2023,manual
##    [3]     chr5      3073      + |      0.44 enhancer     2021,manual
```

```
##    [4]     chr5      4553      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

- Extracting flanking coordinates for each entry in `gr`:

```
gr
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Extract 100bp UPSTREAM of each genomic range, according to range strandness:
flank(gr, 100, start = TRUE)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 1904-2003      + |      0.45 NA                  <NA>
##    [2]     chr4 9874-9973      - |      0.43 promoter     2023,manual
##    [3]     chr5 1843-1942      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4003-4102      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

# ----- Extract 1bp DOWNSTREAM of each genomic range, according to range strandness:
flank(gr, 1, start = FALSE)
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2      7854      + |      0.45 NA                  <NA>
##    [2]     chr4      4481      - |      0.43 promoter     2023,manual
##    [3]     chr5      4204      + |      0.44 enhancer     2021,manual
##    [4]     chr5      5005      + |      0.42 centromere 2019,homology
##    -------
```

```
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

Note how here again, strand information is crucial and correctly leveraged to extract "upstream" or "downstream" flanking regions in agreement with genomic range orientation.

- Several arithmetics operators can also directly work with `GRanges`:

```
gr
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2004-7853      + |      0.45 NA                  <NA>
##    [2]     chr4 4482-9873      - |      0.43 promoter     2023,manual
##    [3]     chr5 1943-4203      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4103-5004      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

gr + 100 # ----- Extend each side of the `GRanges` by a given number of bases
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 1904-7953      + |      0.45 NA                  <NA>
##    [2]     chr4 4382-9973      - |      0.43 promoter     2023,manual
##    [3]     chr5 1843-4303      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4003-5104      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

gr - 200 # ----- Shrink each side of the `GRanges` by a given number of bases
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 2204-7653      + |      0.45 NA                  <NA>
##    [2]     chr4 4682-9673      - |      0.43 promoter     2023,manual
##    [3]     chr5 2143-4003      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4303-4804      + |      0.42 centromere 2019,homology
##    -------
```

```
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths

gr * 1000 # ----- Zoom in by a given factor (effectively decreasing the `GRanges` width by t
##  GRanges object with 4 ranges and 3 metadata columns:
##        seqnames    ranges strand |        GC annotation extended.info
##           <Rle> <IRanges>  <Rle> | <numeric>   <factor>        <list>
##    [1]     chr2 4926-4930      + |      0.45 NA                  <NA>
##    [2]     chr4 7175-7179      - |      0.43 promoter     2023,manual
##    [3]     chr5 3072-3073      + |      0.44 enhancer     2021,manual
##    [4]     chr5 4554-4553      + |      0.42 centromere 2019,homology
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

> **Going further**
>
> To fully grasp how to operate `GRanges` objects, we highly recommend reading the detailed documentation for this class by typing `?GenomicRanges` and `?GenomicRanges::`intra-range-methods``.

2.1.3.2 Inter-range methods

Compared to "intra-range" methods described above, **inter-range** methods involve comparisons *between* ranges **in a single** GRanges object.

> **Note**
>
> Compared to previous section, the result of each function described below depends on the **entire set of ranges** in the input `GRanges` object.

- Computing the "inverse" genomic ranges, i.e. ranges in-between the input ranges:

```
gaps(gr)
##  GRanges object with 3 ranges and 0 metadata columns:
##        seqnames    ranges strand
```

```
##           <Rle> <IRanges>  <Rle>
##    [1]     chr2    1-2003      +
##    [2]     chr4    1-4481      -
##    [3]     chr5    1-1942      +
##    -------
##    seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

- For each entry in a `GRanges`, finding the index of the preceding/following/nearest genomic range:

```
precede(gr)
##  [1] NA NA NA NA

follow(gr)
##  [1] NA NA NA NA

nearest(gr)
##  [1] NA NA  4  3
```

- Computing a coverage over a genome, optionally indicated a "score" column from metadata:

```
coverage(gr, weight = 'GC')
##  RleList of length 3
##  $chr2
##  numeric-Rle of length 7853 with 2 runs
##    Lengths: 2003 5850
##    Values : 0.00 0.45
##
##  $chr4
##  numeric-Rle of length 9873 with 2 runs
##    Lengths: 4481 5392
##    Values : 0.00 0.43
##
##  $chr5
##  numeric-Rle of length 5004 with 4 runs
##    Lengths: 1942 2160  101  801
##    Values : 0.00 0.44 0.86 0.42
```

> ⚠ Going further
>
> To fully grasp how to operate `GRanges` objects, we highly recommend reading the detailed documentation for this class by typing `?GenomicRanges::`inter-range-methods``.

2.1.4 Comparing multiple `GRanges` objects

Genomic analysis typically requires intersection of two sets of genomic ranges, e.g. to find which ranges from one set overlap with those from another set.

In the next examples, we will use two `GRanges`:

- `peaks` represents dummy 8 ChIP-seq peaks

```
peaks <- GRanges(c(
    'chr1:320-418',
    'chr1:512-567',
    'chr1:843-892',
    'chr1:1221-1317',
    'chr1:1329-1372',
    'chr1:1852-1909',
    'chr1:2489-2532',
    'chr1:2746-2790'
))
peaks
##  GRanges object with 8 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr1   320-418      *
##    [2]     chr1   512-567      *
##    [3]     chr1   843-892      *
##    [4]     chr1 1221-1317      *
##    [5]     chr1 1329-1372      *
##    [6]     chr1 1852-1909      *
##    [7]     chr1 2489-2532      *
##    [8]     chr1 2746-2790      *
##    -------
```

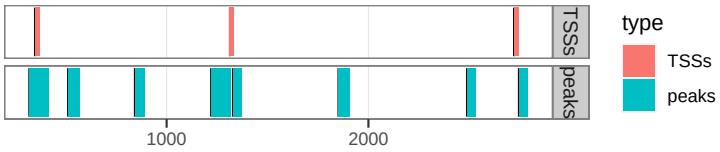
```
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- `TSSs` represents dummy 3 gene promoters ($\pm$ 10bp around the TSS)

```
genes <- GRanges(c(
    'chr1:358-1292:+',
    'chr1:1324-2343:+',
    'chr1:2732-2751:+'
))
TSSs <- resize(genes, width = 1, fix = 'start') + 10
TSSs
##  GRanges object with 3 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr1   348-368      +
##    [2]     chr1 1314-1334      +
##    [3]     chr1 2722-2742      +
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Let's see how they overlap by plotting them:

```
library(ggplot2)
peaks$type <- 'peaks'
TSSs$type <- 'TSSs'
ggplot() +
    ggbio::geom_rect(c(peaks, TSSs), aes(fill = type), facets = type~.) +
    ggbio::theme_alignment() +
    coord_fixed(ratio = 300)
##  Registered S3 method overwritten by 'GGally':
##    method from
##    +.gg   ggplot2
##  Warning: The `facets` argument of `facet_grid()` is deprecated as of ggplot2 2.2.0.
##  i Please use the `rows` argument instead.
##  i The deprecated feature was likely used in the ggbio package.
##    Please report the issue at <https://github.com/lawremi/ggbio/issues>.
##  Scale for y is already present.
##  Adding another scale for y, which will replace the existing scale.
```



2.1.4.1 Finding overlaps between two `GRanges` sets

- Finding overlaps between a query and a subject

In our case, we want to identify which ChIP-seq peaks overlap with a TSS: the query is the set of peaks and the subject is the set of TSSs.

`findOverlaps` returns a `Hits` object listing which `query` ranges overlap with which `subject` ranges.

```
ov <- findOverlaps(query = peaks, subject = TSSs)
ov
##  Hits object with 3 hits and 0 metadata columns:
##         queryHits subjectHits
##         <integer>   <integer>
##     [1]         1           1
##     [2]         4           2
##     [3]         5           2
##     -------
##     queryLength: 8 / subjectLength: 3
```

The `Hits` output clearly describes what overlaps with what:

- The query (peak) `#1` overlaps with subject (TSS) `#1`
- The query (peak) `#5` overlaps with subject (TSS) `#2`

> **Note**
>
> Because no other query index or subject index is listed in the `ov` output, none of the remaining ranges from `query` overlap with ranges from `subject`.

- Subsetting by overlaps between a query and a subject

To directly **subset** ranges from `query` overlapping with ranges from a `subject` (e.g. to only keep *peaks* overlapping a *TSS*), we can use the `subsetByOverlaps` function. The output of `subsetByOverlaps` is a subset of the original `GRanges` object provided as a `query`, with retained ranges being unmodified.

```
subsetByOverlaps(peaks, TSSs)
##  GRanges object with 3 ranges and 1 metadata column:
##        seqnames    ranges strand |        type
##           <Rle> <IRanges>  <Rle> | <character>
##    [1]     chr1   320-418      * |       peaks
##    [2]     chr1 1221-1317      * |       peaks
##    [3]     chr1 1329-1372      * |       peaks
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- Counting overlaps between a query and a subject

Finally, the `countOverlaps` is used to count, for each range in a `query`, how many ranges in the `subject` it overlaps with.

```
countOverlaps(query = peaks, subject = TSSs)
##  [1] 1 0 0 1 1 0 0 0
```

> **Note**
>
> Note that which `GRanges` goes in `query` or `subject` is crucial! Counting **for each peak, the number of TSSs it overlaps with** is very different from **for each TSS, how many peaks it overlaps with**.
> In our case example, it would also be informative to count how many peaks overlap with each TSS, so we'd need to

swap `query` and `subject`:

```
countOverlaps(query = TSSs, subject = peaks)
##  [1] 1 2 0
```

We can add these counts to the original `query` object:

```
TSSs$n_peaks <- countOverlaps(query = TSSs, subject = peaks)
TSSs
##  GRanges object with 3 ranges and 2 metadata columns:
##        seqnames    ranges strand |        type   n_peaks
##           <Rle> <IRanges>  <Rle> | <character> <integer>
##    [1]     chr1   348-368      + |        TSSs         1
##    [2]     chr1 1314-1334      + |        TSSs         2
##    [3]     chr1 2722-2742      + |        TSSs         0
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- `%over%`, `%within%`, `%outside%` : handy operators

Handy operators exist that return logical vectors (same length as the `query`). They essentially are short-hands for specific `findOverlaps()` cases.

`<query> %over% <subject>`:

```
peaks %over% TSSs
##  [1]  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE

peaks[peaks %over% TSSs] # ----- Equivalent to `subsetByOverlaps(peaks, TSSs)`
##  GRanges object with 3 ranges and 1 metadata column:
##        seqnames    ranges strand |        type
##           <Rle> <IRanges>  <Rle> | <character>
##    [1]     chr1   320-418      * |       peaks
##    [2]     chr1 1221-1317      * |       peaks
##    [3]     chr1 1329-1372      * |       peaks
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

`<query> %within% <subject>`:

```
peaks %within% TSSs
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

TSSs %within% peaks
##  [1]  TRUE FALSE FALSE
```

`<query> %outside% <subject>`:

```
peaks %outside% TSSs
##  [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
```

> ⚠ Going further
>
> To fully grasp how to find overlaps between `GRanges` objects, we highly recommend reading the detailed documentation by typing `?IRanges::`findOverlaps-methods``.

2.1.4.2 Find nearest range from a subject for each range in a query

`*Overlaps` methods are not always enough to match a `query` to a `subject`. For instance, some peaks in the `query` might be very near to some TSSs in the `subject`, but not quite overlapping.

```
peaks[8]
##  GRanges object with 1 range and 1 metadata column:
##        seqnames    ranges strand |        type
##           <Rle> <IRanges>  <Rle> | <character>
##    [1]     chr1 2746-2790      * |       peaks
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths

TSSs[3]
##  GRanges object with 1 range and 2 metadata columns:
##        seqnames    ranges strand |        type   n_peaks
##           <Rle> <IRanges>  <Rle> | <character> <integer>
##    [1]     chr1 2722-2742      + |        TSSs         0
##    -------
```

```
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- `nearest()`

Rather than finding the *overlapping* range in a `subject` for each range in a `query`, we can find the `nearest` range.

For each range in the `query`, this returns the *index* of the range in the `subject` to which the `query` is the nearest.

```
nearest(peaks, TSSs)
##  [1] 1 1 2 2 2 2 3 3

TSSs[nearest(peaks, TSSs)]
##  GRanges object with 8 ranges and 2 metadata columns:
##        seqnames    ranges strand |        type   n_peaks
##           <Rle> <IRanges>  <Rle> | <character> <integer>
##    [1]     chr1   348-368      + |        TSSs         1
##    [2]     chr1   348-368      + |        TSSs         1
##    [3]     chr1 1314-1334      + |        TSSs         2
##    [4]     chr1 1314-1334      + |        TSSs         2
##    [5]     chr1 1314-1334      + |        TSSs         2
##    [6]     chr1 1314-1334      + |        TSSs         2
##    [7]     chr1 2722-2742      + |        TSSs         0
##    [8]     chr1 2722-2742      + |        TSSs         0
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- `distance()`

Alternatively, one can simply ask to calculate the `distanceToNearest` between ranges in a `query` and ranges in a `subject`.

```
distanceToNearest(peaks, TSSs)
##  Hits object with 8 hits and 1 metadata column:
##        queryHits subjectHits |  distance
##        <integer>   <integer> | <integer>
##    [1]         1           1 |         0
##    [2]         2           1 |       143
##    [3]         3           2 |       421
```

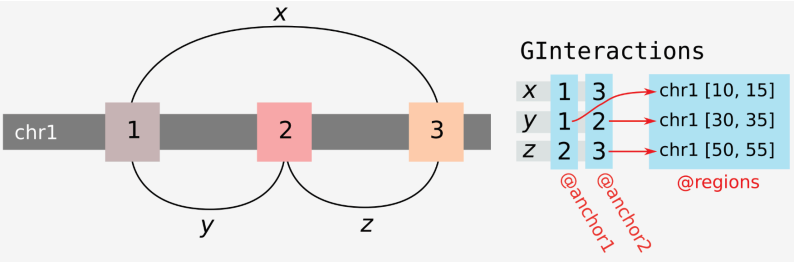
```
##     [4]         4           2 |         0
##     [5]         5           2 |         0
##     [6]         6           2 |       517
##     [7]         7           3 |       189
##     [8]         8           3 |         3
##     -------
##     queryLength: 8 / subjectLength: 3
```

```
peaks$distance_to_nearest_TSS <- mcols(distanceToNearest(peaks, TSSs))$distance
```

Note how close from a TSS the 8th peak was. It could be worth considering this as an overlap!

2.2 `GInteractions` class

`GRanges` describe genomic ranges and hence are of general use to study 1D genome organization. To study chromatin interactions, we need a way to link pairs of `GRanges`. This is exactly what the `GInteractions` class does. This data structure is defined in the `InteractionSet` package and has been published in the 2016 paper by `Lun et al.` (@Lun_2016).



2.2.1 Building a `GInteractions` object from scratch

Let's first define two parallel `GRanges` objects (i.e. two `GRanges` of same length). Each `GRanges` will contain 5 ranges.

```
gr_first <- GRanges(c(
    'chr1:1-100',
    'chr1:1001-2000',
```

```
    'chr1:5001-6000',
    'chr1:8001-9000',
    'chr1:7001-8000'
))
gr_second <- GRanges(c(
    'chr1:1-100',
    'chr1:3001-4000',
    'chr1:8001-9000',
    'chr1:7001-8000',
    'chr2:13000-14000'
))
```

Because these two `GRanges` objects are of same length (5), one can "bind" them together by using the `GInteractions`function. This effectively associate each entry from one `GRanges` to the entry aligned in the other `GRanges` object.

```
library(InteractionSet)
gi <- GInteractions(gr_first, gr_second)
gi
##  GInteractions object with 5 interactions and 0 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2
##            <Rle> <IRanges>         <Rle>   <IRanges>
##    [1]      chr1     1-100 ---      chr1       1-100
##    [2]      chr1 1001-2000 ---      chr1   3001-4000
##    [3]      chr1 5001-6000 ---      chr1   8001-9000
##    [4]      chr1 8001-9000 ---      chr1   7001-8000
##    [5]      chr1 7001-8000 ---      chr2 13000-14000
##    -------
##    regions: 7 ranges and 0 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

The way `GInteractions` objects are printed in an R console mimics that of `GRanges`, but pairs two "ends" (a.k.a. *anchors*) of an interaction together, each end being represented as a separate `GRanges` range.

- Note that it is possible to have interactions joining two identical anchors.

```
gi[1]
##  GInteractions object with 1 interaction and 0 metadata columns:
##        seqnames1   ranges1     seqnames2   ranges2
##            <Rle> <IRanges>         <Rle> <IRanges>
##    [1]      chr1     1-100 ---      chr1     1-100
##    -------
##    regions: 7 ranges and 0 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

- It is also technically possible (though not advised) to have interactions for which the "first" end is located after the "second" end along the chromosome.

```
gi[4]
##  GInteractions object with 1 interaction and 0 metadata columns:
##        seqnames1   ranges1     seqnames2   ranges2
##            <Rle> <IRanges>         <Rle> <IRanges>
##    [1]      chr1 8001-9000 ---      chr1 7001-8000
##    -------
##    regions: 7 ranges and 0 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

- Finally, it is possible to define inter-chromosomal interactions (a.k.a. trans interactions).

```
gi[5]
##  GInteractions object with 1 interaction and 0 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2
##            <Rle> <IRanges>         <Rle>   <IRanges>
##    [1]      chr1 7001-8000 ---      chr2 13000-14000
##    -------
##    regions: 7 ranges and 0 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

2.2.2 `GInteractions` specific slots

Compared to `GRanges`, extra slots are available for `GInteractions` objects, e.g. `anchors` and `regions`.

2.2.2.1 Anchors

"Anchors" of a single genomic interaction refer to the two ends of this interaction. These anchors can be extracted from a `GInteractions` object using the `anchors()` function. This outputs a list of two `GRanges`, the first corresponding to the "left" end of interactions (when printed to the console) and the second corresponding to the "right" end of interactions (when printed to the console).

```
# ----- This extracts the two sets of anchors ("first" and "second") from a GInteractions ob
anchors(gi)
##  $first
##  GRanges object with 5 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr1     1-100      *
##    [2]     chr1 1001-2000      *
##    [3]     chr1 5001-6000      *
##    [4]     chr1 8001-9000      *
##    [5]     chr1 7001-8000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
##
##  $second
##  GRanges object with 5 ranges and 0 metadata columns:
##        seqnames      ranges strand
##           <Rle>   <IRanges>  <Rle>
##    [1]     chr1       1-100      *
##    [2]     chr1   3001-4000      *
##    [3]     chr1   8001-9000      *
##    [4]     chr1   7001-8000      *
##    [5]     chr2 13000-14000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

# ----- We can query for the "first" or "second" set of anchors directly
anchors(gi, "first")
##  GRanges object with 5 ranges and 0 metadata columns:
##        seqnames    ranges strand
```

```
##           <Rle> <IRanges>  <Rle>
##    [1]     chr1     1-100      *
##    [2]     chr1 1001-2000      *
##    [3]     chr1 5001-6000      *
##    [4]     chr1 8001-9000      *
##    [5]     chr1 7001-8000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

anchors(gi, "second")
##  GRanges object with 5 ranges and 0 metadata columns:
##        seqnames      ranges strand
##           <Rle>   <IRanges>  <Rle>
##    [1]     chr1       1-100      *
##    [2]     chr1   3001-4000      *
##    [3]     chr1   8001-9000      *
##    [4]     chr1   7001-8000      *
##    [5]     chr2 13000-14000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

2.2.2.2 Regions

"Regions" of a *set* of interactions refer to the universe of unique anchors represented in a set of interactions. Therefore, the length of the `regions` can only be equal to or strictly lower than twice the length of `anchors`.

The `regions` function returns the regions associated with a `GInteractions` object, stored as a `GRanges` object.

```
regions(gi)
##  GRanges object with 7 ranges and 0 metadata columns:
##        seqnames      ranges strand
##           <Rle>   <IRanges>  <Rle>
##    [1]     chr1       1-100      *
##    [2]     chr1   1001-2000      *
##    [3]     chr1   3001-4000      *
##    [4]     chr1   5001-6000      *
```

```
##    [5]     chr1   7001-8000      *
##    [6]     chr1   8001-9000      *
##    [7]     chr2 13000-14000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

length(regions(gi))
##  [1] 7

length(anchors(gi, "first"))
##  [1] 5
```

2.2.3 `GInteractions` methods

`GInteractions` behave as an extension of `GRanges`. For this reason, many methods that work with `GRanges` will work seamlessly with `GInteractions`.

2.2.3.1 Metadata

One can add metadata columns directly to a `GInteractions` object.

```
mcols(gi)
##  DataFrame with 5 rows and 0 columns
mcols(gi) <- data.frame(
    idx = seq(1, length(gi)),
    type = c("cis", "cis", "cis", "trans", "cis")
)
gi
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [5]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
```

```
##    -------
##    regions: 7 ranges and 0 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

gi$type
##  [1] "cis"   "cis"   "cis"   "trans" "cis"
```

Importantly, metadata columns can also be directly added to **regions** of a `GInteractions` object, since these `regions` are a `GRanges` object themselves!

```
regions(gi)
##  GRanges object with 7 ranges and 0 metadata columns:
##        seqnames      ranges strand
##           <Rle>   <IRanges>  <Rle>
##    [1]     chr1       1-100      *
##    [2]     chr1   1001-2000      *
##    [3]     chr1   3001-4000      *
##    [4]     chr1   5001-6000      *
##    [5]     chr1   7001-8000      *
##    [6]     chr1   8001-9000      *
##    [7]     chr2 13000-14000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
regions(gi)$binID <- seq_along(regions(gi))
regions(gi)$type <- c("P", "P", "P", "E", "E", "P", "P")
regions(gi)
##  GRanges object with 7 ranges and 2 metadata columns:
##        seqnames      ranges strand |     binID        type
##           <Rle>   <IRanges>  <Rle> | <integer> <character>
##    [1]     chr1       1-100      * |         1           P
##    [2]     chr1   1001-2000      * |         2           P
##    [3]     chr1   3001-4000      * |         3           P
##    [4]     chr1   5001-6000      * |         4           E
##    [5]     chr1   7001-8000      * |         5           E
##    [6]     chr1   8001-9000      * |         6           P
##    [7]     chr2 13000-14000      * |         7           P
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

2.2.3.2 Sorting `GInteractions`

The `sort` function works seamlessly with `GInteractions` objects. It sorts the interactions using a similar approach to that performed by `pairtools sort ...` for disk-stored `.pairs` files, sorting on the "first" anchor first, then for interactions with the same "first" anchors, sorting on the "second" anchor.

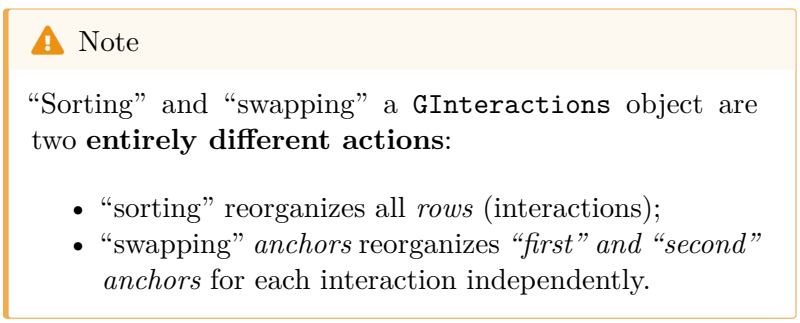
```
gi
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [5]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

sort(gi)
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    [5]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

2.2.3.3 Swapping `GInteractions` anchors

For an individual interaction contained in a `GInteractions` object, the "first" and "second" anchors themselves can be sorted as well. This is called "pairs swapping", and it is performed similarly to `pairtools flip ...` for disk-stored `.pairs` files.

This ensures that interactions, when represented as a contact matrix, generate an upper-triangular matrix.

```
gi
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [5]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

swapAnchors(gi)
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 7001-8000 ---      chr1   8001-9000 |         4       trans
##    [5]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```



Note

"Sorting" and "swapping" a `GInteractions` object are two **entirely different actions**:

- "sorting" reorganizes all *rows* (interactions);
- "swapping" *anchors* reorganizes *"first" and "second" anchors* for each interaction independently.

2.2.3.4 `GInteractions` distance method

"Distance", when applied to genomic interactions, typically refers to the genomic distance between the two anchors of a single interaction. For `GInteractions`, this is computed using the `pairdist` function.

```
gi
##  GInteractions object with 5 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1     1-100 ---      chr1       1-100 |         1         cis
##    [2]      chr1 1001-2000 ---      chr1   3001-4000 |         2         cis
##    [3]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [4]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [5]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths

pairdist(gi)
##  [1]    0 2000 3000 1000   NA
```

Note that for "trans" inter-chromosomal interactions, i.e. interactions with anchors on different chromosomes, the notion of genomic distance is meaningless and for this reason, `pairdist` returns a `NA` value.

The `type` argument of the `pairdist()` function can be tweaked to specify which type of "distance" should be computed:

- `mid`: The distance between the midpoints of the two regions (rounded down to the nearest integer) is returned (Default).
- `gap`: The length of the gap between the closest points of the two regions is computed - negative lengths are returned for overlapping regions, indicating the length of the overlap.
- `span`: The distance between the furthermost points of the two regions is computed.

- `diag`: The difference between the anchor indices is returned. This corresponds to a diagonal on the interaction space when bins are used in the 'regions' slot of 'x'.

2.2.3.5 `GInteractions` overlap methods

"Overlaps" for genomic interactions could be computed in different contexts:

- Case 1: Overlap between any of the two anchors of an interaction with a genomic range
- Case 2: Overlap between anchors of an interaction with anchors of another interaction
- Case 3: Spanning of the interaction "across" a genomic range

Case 1: Overlap between any of the two anchors of an interaction with a genomic range

This is the default behavior of `findOverlaps` when providing a `GInteractions` object as `query` and a `GRanges` as a `subject`.

```
gr <- GRanges(c("chr1:7501-7600", "chr1:8501-8600"))
findOverlaps(query = gi, subject = gr)
##  Hits object with 4 hits and 0 metadata columns:
##         queryHits subjectHits
##         <integer>   <integer>
##     [1]         3           2
##     [2]         4           1
##     [3]         4           2
##     [4]         5           1
##     -------
##     queryLength: 5 / subjectLength: 2

countOverlaps(gi, gr)
##  [1] 0 0 1 2 1

subsetByOverlaps(gi, gr)
##  GInteractions object with 3 interactions and 2 metadata columns:
##         seqnames1   ranges1     seqnames2     ranges2 |       idx        type
```

```
##           <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [2]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [3]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

Here again, the order matters!

```
countOverlaps(gr, gi)
##  [1] 2 2
```

And again, the `%over%` operator can be used here:

```
gi %over% gr
##  [1] FALSE FALSE  TRUE  TRUE  TRUE

gi[gi %over% gr] # ----- Equivalent to `subsetByOverlaps(gi, gr)`
##  GInteractions object with 3 interactions and 2 metadata columns:
##        seqnames1   ranges1     seqnames2     ranges2 |       idx        type
##            <Rle> <IRanges>         <Rle>   <IRanges> | <integer> <character>
##    [1]      chr1 5001-6000 ---      chr1   8001-9000 |         3         cis
##    [2]      chr1 8001-9000 ---      chr1   7001-8000 |         4       trans
##    [3]      chr1 7001-8000 ---      chr2 13000-14000 |         5         cis
##    -------
##    regions: 7 ranges and 2 metadata columns
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

Case 2: Overlap between anchors of an interaction with anchors of another interaction

This slightly different scenario involves overlapping two sets of interactions, to see whether any interaction in `Set-1` has its two anchors overlapping anchors from an interaction in `Set-2`.

```
gi2 <- GInteractions(
    GRanges("chr1:1081-1090"),
    GRanges("chr1:3401-3501")
)
```

```
gi %over% gi2
##  [1] FALSE  TRUE FALSE FALSE FALSE
```

Note that both anchors of an interaction from a `query` have to overlap to a pair of anchors of a single interaction from a `subject` with this method!

```
gi3 <- GInteractions(
    GRanges("chr1:1-1000"),
    GRanges("chr1:3401-3501")
)
gi %over% gi3
##  [1] FALSE FALSE FALSE FALSE FALSE
```

Case 3 : Spanning of the interaction "accross" a genomic range

This requires a bit of wrangling, to mimic an overlap between two `GRanges` objects:

```
gi <- swapAnchors(gi) # ----- Make sure anchors are correctly sorted
gi <- sort(gi) # ----- Make sure interactions are correctly sorted
gi <- gi[!is.na(pairdist(gi))] # ----- Remove inter-chromosomal interactions
spanning_gi <- GRanges(
    seqnames = seqnames(anchors(gi)[[1]]),
    ranges = IRanges(
        start(anchors(gi)[[1]]),
        end(anchors(gi)[[2]])
    )
)
spanning_gi
##  GRanges object with 4 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]     chr1     1-100      *
##    [2]     chr1 1001-4000      *
##    [3]     chr1 5001-9000      *
##    [4]     chr1 7001-9000      *
##    -------
##    seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
spanning_gi %over% gr
##  [1] FALSE FALSE  TRUE  TRUE
```

> ⚠ Going further
>
> A detailed manual of overlap methods available for `GInteractions` object can be read by typing `?`Interaction-overlaps`` in R.

2.3 `ContactFile` class

Hi-C contacts can be stored in four different formats (see previous chapter):

- As a `.(m)cool` matrix (multi-scores, multi-resolution, indexed)
- As a `.hic` matrix (multi-scores, multi-resolution, indexed)
- As a HiC-pro derived matrix (single-score, single-resolution, non-indexed)
- Unbinned, Hi-C contacts can be stored in `.pairs` files

2.3.1 Accessing example Hi-C files

Example contact files can be downloaded using `HiContactsData` function.

```
library(HiContactsData)
coolf <- HiContactsData('yeast_wt', 'mcool')
```

This fetches files from the cloud, download them locally and returns the path of the local file.

```
coolf
##                                                    EH7702
##  "/root/.cache/R/ExperimentHub/174688ce76a_7752"
```

Similarly, example files are available for other file formats:

```
hicf <- HiContactsData('yeast_wt', 'hic')
hicpromatrixf <- HiContactsData('yeast_wt', 'hicpro_matrix')
hicproregionsf <- HiContactsData('yeast_wt', 'hicpro_bed')
pairsf <- HiContactsData('yeast_wt', 'pairs.gz')
```

We can even check the content of some of these files to make sure they are actually what they are:

```
# ---- HiC-Pro generates a tab-separated `regions.bed` file
readLines(hicproregionsf, 25)
##   [1] "I\t0\t1000"      "I\t1000\t2000"   "I\t2000\t3000"   "I\t3000\t4000"
##   [5] "I\t4000\t5000"   "I\t5000\t6000"   "I\t6000\t7000"   "I\t7000\t8000"
##   [9] "I\t8000\t9000"   "I\t9000\t10000"  "I\t10000\t11000" "I\t11000\t12000"
##  [13] "I\t12000\t13000" "I\t13000\t14000" "I\t14000\t15000" "I\t15000\t16000"
##  [17] "I\t16000\t17000" "I\t17000\t18000" "I\t18000\t19000" "I\t19000\t20000"
##  [21] "I\t20000\t21000" "I\t21000\t22000" "I\t22000\t23000" "I\t23000\t24000"
##  [25] "I\t24000\t25000"

# ---- Pairs are also tab-separated
readLines(pairsf, 25)
##   [1] "## pairs format v1.0"
##   [2] "#sorted: chr1-pos1-chr2-pos2"
##   [3] "#columns: readID chr1 pos1 chr2 pos2 strand1 strand2 frag1 frag2"
##   [4] "#chromsize: I 230218"
##   [5] "#chromsize: II 813184"
##   [6] "#chromsize: III 316620"
##   [7] "#chromsize: IV 1531933"
##   [8] "#chromsize: V 576874"
##   [9] "#chromsize: VI 270161"
##  [10] "#chromsize: VII 1090940"
##  [11] "#chromsize: VIII 562643"
##  [12] "#chromsize: IX 439888"
##  [13] "#chromsize: X 745751"
##  [14] "#chromsize: XI 666816"
##  [15] "#chromsize: XII 1078177"
##  [16] "#chromsize: XIII 924431"
##  [17] "#chromsize: XIV 784333"
##  [18] "#chromsize: XV 1091291"
##  [19] "#chromsize: XVI 948066"
##  [20] "#chromsize: Mito 85779"
```

```
##  [21] "NS500150:527:HHGYNBGXF:3:21611:19085:3986\tII\t105\tII\t48548\t+\t-\t1358\t1681"
##  [22] "NS500150:527:HHGYNBGXF:4:13604:19734:2406\tII\t113\tII\t45003\t-\t+\t1358\t1658"
##  [23] "NS500150:527:HHGYNBGXF:2:11108:25178:11036\tII\t119\tII\t687251\t-\t+\t1358\t5550"
##  [24] "NS500150:527:HHGYNBGXF:1:22301:8468:1586\tII\t160\tII\t26124\t+\t-\t1358\t1510"
##  [25] "NS500150:527:HHGYNBGXF:4:23606:24037:2076\tII\t169\tII\t39052\t+\t+\t1358\t1613"
```

2.3.2 `ContactFile` fundamentals

A `ContactFile` object **establishes a connection with a disk-stored Hi-C file** (e.g. a `.cool` file, or a `.pairs` file, …). `ContactFile` classes are defined in the `HiCExperiment` package.

`ContactFile`s come in four different flavors:

- `CoolFile`: connection to a `.(m)cool` file
- `HicFile`: connection to a `.hic` file
- `HicproFile`: connection to output files generated by HiC-Pro
- `PairsFile`: connection to a `.pairs` file

To create each flavor of `ContactFile`, one can use the corresponding function:

```
library(HiCExperiment)

# ----- This creates a connection to a `.(m)cool` file (path stored in `coolf`)
CoolFile(coolf)
##  CoolFile object
##  .mcool file: /root/.cache/R/ExperimentHub/174688ce76a_7752
##  resolution: 1000
##  pairs file:
##  metadata(0):

# ----- This creates a connection to a `.hic` file (path stored in `hicf`)
HicFile(hicf)
##  HicFile object
##  .hic file: /root/.cache/R/ExperimentHub/17460f12195_7836
##  resolution: 1000
##  pairs file:
```

```
##  metadata(0):

# ----- This creates a connection to output files from HiC-Pro
HicproFile(hicpromatrixf, hicproregionsf)
##  HicproFile object
##  HiC-Pro files:
##    $ matrix:   /root/.cache/R/ExperimentHub/1745c1383e1_7837
##    $ regions:  /root/.cache/R/ExperimentHub/17439b9b892_7838
##  resolution: 1000
##  pairs file:
##  metadata(0):

# ----- This creates a connection to a pairs file
PairsFile(pairsf)
##  PairsFile object
##  resource: /root/.cache/R/ExperimentHub/174ff8a7b2_7753
```

2.3.3 `ContactFile` slots

Several "slots" (i.e. pieces of information) are attached to a `ContactFile` object:

- The path to the disk-stored contact matrix;
- The active resolution (by default, the finest resolution available in a multi-resolution contact matrix);
- Optionally, the path to a matching `pairs` file (see below);
- Some metadata.

Slots of a `CoolFile` object can be accessed as follow:

```
cf <- CoolFile(coolf)
cf
##  CoolFile object
##  .mcool file: /root/.cache/R/ExperimentHub/174688ce76a_7752
##  resolution: 1000
##  pairs file:
##  metadata(0):

resolution(cf)
```

```
##  [1] 1000

pairsFile(cf)
##  NULL

metadata(cf)
##  list()
```

> ⚠ Important!
>
> `ContactFile` objects are only **connections** to a disk-stored HiC file. Although metadata is available, they do not contain actual data!

2.3.4 `ContactFile` methods

Two useful methods are available for `ContactFile`s:

- `availableResolutions` checks which resolutions are available in a `ContactFile`.

```
availableResolutions(cf)
##  resolutions(5): 1000 2000 4000 8000 16000
##
```

- `availableChromosomes` checks which chromosomes are available in a `ContactFile`, along with their length.

```
availableChromosomes(cf)
##  Seqinfo object with 16 sequences from an unspecified genome:
##    seqnames seqlengths isCircular genome
##    I            230218       <NA>   <NA>
##    II           813184       <NA>   <NA>
##    III          316620       <NA>   <NA>
##    IV          1531933       <NA>   <NA>
##    V            576874       <NA>   <NA>
##    ...             ...        ...    ...
##    XII         1078177       <NA>   <NA>
##    XIII         924431       <NA>   <NA>
```

```
##    XIV          784333       <NA>   <NA>
##    XV          1091291       <NA>   <NA>
##    XVI          948066       <NA>   <NA>
```

2.4 `HiCExperiment` class

Based on the previous sections, we have different Bioconductor classes relevant for Hi-C:

- `GInteractions` which can be used to represent genomic interactions in R
- `ContactFile`s which can be used to establish a connection with disk-stored Hi-C files

`HiCExperiment` objects are created when *parsing* a `ContactFile` in R. The `HiCExperiment` class reads a `ContactFile` in memory and store genomic interactions as `GInteractions`. The `HiCExperiment` class is, quite obviously, defined in the `HiCExperiment` package.

2.4.1 Creating a `HiCExperiment` object

2.4.1.1 Importing a `ContactFile`

In practice, to create a `HiCExperiment` object from a `ContactFile`, one can use the `import` method.

> **!** Caution
>
> - Creating a `HiCExperiment` object means ***importing data from a Hi-C matrix*** (e.g. from a `ContactFile`) in memory in R.
>
> - Creating a `HiCExperiment` object from large disk-stored contact matrices can potentially take a long time.

```
cf <- CoolFile(coolf)
hic <- import(cf)
hic
##  `HiCExperiment` object with 8,757,906 contacts over 12,079 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "whole genome"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 1000
##  interactions: 2945692
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

Printing a `HiCExperiment` to the console will not reveal the actual data stored in the object (it would most likely crash your R session!). Instead, it gives a **summary of the data** stored in the object:

- The `fileName`, i.e. the path to the disk-stored data file
- The `focus`, i.e. the genomic location for which data has been imported (in the example above, `"whole genome"` implies that all the data has been imported in R)
- `resolutions` available in the disk-stored data file (this will be identical to `availableResolutions(cf)`)
- `active resolution` indicates at which resolution the data is currently imported
- `interactions` refers to the actual `GInteractions` imported in R and "hidden" (for now!) in the `HiCExperiment` object
- `scores` refer to different interaction frequency estimates. These can be raw `counts`, `balanced` (if the contact matrix has been previously normalized), or whatever score the end-user want to attribute to each interaction (e.g. ratio of counts between two Hi-C maps, …)
- `topologicalFeatures` is a `list` of `GRanges` or `GInteractions` objects to describe important topological features.
- `pairsFile` is a pointer to an optional disk-stored `.pairs` file from which the contact matrix has been created. This

is often useful to estimate some Hi-C metrics.
- `metadata` is a `list` to further describe the experiment.

These pieces of information are called `slots`. They can be directly accessed using `getter` functions, bearing the same name than the slot.

```
fileName(hic)
##  [1] "/root/.cache/R/ExperimentHub/174688ce76a_7752"

focus(hic)
##  NULL

resolutions(hic)
##  [1]  1000  2000  4000  8000 16000

resolution(hic)
##  [1] 1000

interactions(hic)
##  GInteractions object with 2945692 interactions and 4 metadata columns:
##              seqnames1       ranges1     seqnames2       ranges2 |   bin_id1
##                  <Rle>     <IRanges>         <Rle>     <IRanges> | <numeric>
##          [1]         I        1-1000 ---         I        1-1000 |         0
##          [2]         I        1-1000 ---         I     1001-2000 |         0
##          [3]         I        1-1000 ---         I     2001-3000 |         0
##          [4]         I        1-1000 ---         I     3001-4000 |         0
##          [5]         I        1-1000 ---         I     4001-5000 |         0
##          ...       ...           ... ...       ...           ... .       ...
##    [2945688]       XVI 940001-941000 ---       XVI 942001-943000 |     12070
##    [2945689]       XVI 940001-941000 ---       XVI 943001-944000 |     12070
##    [2945690]       XVI 941001-942000 ---       XVI 941001-942000 |     12071
##    [2945691]       XVI 941001-942000 ---       XVI 942001-943000 |     12071
##    [2945692]       XVI 941001-942000 ---       XVI 943001-944000 |     12071
##                bin_id2     count  balanced
##              <numeric> <numeric> <numeric>
##          [1]         0        15 0.0663491
##          [2]         1        21 0.1273505
##          [3]         2        21 0.0738691
##          [4]         3        38 0.0827051
```

```
##          [5]         4        17 0.0591984
##          ...       ...       ...       ...
##    [2945688]     12072        11 0.0575550
##    [2945689]     12073         1       NaN
##    [2945690]     12071        74 0.0504615
##    [2945691]     12072        39 0.1624599
##    [2945692]     12073         1       NaN
##    -------
##    regions: 12079 ranges and 4 metadata columns
##    seqinfo: 16 sequences from an unspecified genome

scores(hic)
##  List of length 2
##  names(2): count balanced

topologicalFeatures(hic)
##  List of length 4
##  names(4): compartments borders loops viewpoints

pairsFile(hic)
##  NULL

metadata(hic)
##  list()
```

`import` also works for other types of `ContactFile` (`HicFile`, `HicproFile`, `PairsFile`), e.g.

- For `HicFile` and `HicproFile`, `import` seamlessly returns a `HiCExperiment` as well:

```
hf <- HicFile(hicf)
hic <- import(hf)
hic
##  `HiCExperiment` object with 13,681,280 contacts over 12,165 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/17460f12195_7836"
##  focus: "whole genome"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 1000
```

```
##  interactions: 2965693
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- For `PairsFile`, the returned object is a representation of Hi-C "pairs" in R, i.e. `GInteractions`

```
pf <- PairsFile(pairsf)
pairs <- import(pf)
pairs
##  GInteractions object with 471364 interactions and 3 metadata columns:
##              seqnames1   ranges1     seqnames2   ranges2 |     frag1     frag2
##                  <Rle> <IRanges>         <Rle> <IRanges> | <numeric> <numeric>
##         [1]        II       105 ---        II     48548 |      1358      1681
##         [2]        II       113 ---        II     45003 |      1358      1658
##         [3]        II       119 ---        II    687251 |      1358      5550
##         [4]        II       160 ---        II     26124 |      1358      1510
##         [5]        II       169 ---        II     39052 |      1358      1613
##         ...       ...       ... ...       ...       ... .       ...       ...
##    [471360]        II    808605 ---        II    809683 |      6316      6320
##    [471361]        II    808609 ---        II    809917 |      6316      6324
##    [471362]        II    808617 ---        II    809506 |      6316      6319
##    [471363]        II    809447 ---        II    809685 |      6319      6321
##    [471364]        II    809472 ---        II    809675 |      6319      6320
##              distance
##             <integer>
##         [1]     48443
##         [2]     44890
##         [3]    687132
##         [4]     25964
##         [5]     38883
##         ...       ...
##    [471360]      1078
##    [471361]      1308
##    [471362]       889
##    [471363]       238
##    [471364]       203
##    -------
```

```
##    regions: 549331 ranges and 0 metadata columns
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

2.4.1.2 Customizing the `import`

To reduce the `import` to only parse the data that is relevant to the study, two arguments can be passed to `import`, along with a `ContactFile`.

> ⚠ Key `import` arguments:
>
> - `focus`: This can be used to **only parse data for a specific genomic location**.
> - `resolution`: This can be used to choose which resolution to parse the contact matrix at (this is ignored if the `ContactFile` is not multi-resolution, e.g. `.cool` or HiC-Pro generated matrices)

- Import interactions within a single chromosome:

```
hic <- import(cf, focus = 'II', resolution = 2000)

regions(hic) # ---- `regions()` work on `HiCExperiment` the same way than on `GInteractions`
##  GRanges object with 407 ranges and 4 metadata columns:
##                      seqnames        ranges strand |    bin_id    weight   chr
##                         <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##           II_1_2000       II        1-2000      * |       116       NaN    II
##        II_2001_4000       II     2001-4000      * |       117       NaN    II
##        II_4001_6000       II     4001-6000      * |       118       NaN    II
##        II_6001_8000       II     6001-8000      * |       119       NaN    II
##       II_8001_10000       II    8001-10000      * |       120 0.0461112    II
##                 ...      ...           ...    ... .       ...       ...   ...
##    II_804001_806000       II 804001-806000      * |       518 0.0493107    II
##    II_806001_808000       II 806001-808000      * |       519 0.0611355    II
##    II_808001_810000       II 808001-810000      * |       520       NaN    II
##    II_810001_812000       II 810001-812000      * |       521       NaN    II
##    II_812001_813184       II 812001-813184      * |       522       NaN    II
##                      center
```

```
##                     <integer>
##          II_1_2000      1000
##       II_2001_4000      3000
##       II_4001_6000      5000
##       II_6001_8000      7000
##      II_8001_10000      9000
##                ...       ...
##   II_804001_806000    805000
##   II_806001_808000    807000
##   II_808001_810000    809000
##   II_810001_812000    811000
##   II_812001_813184    812592
##   -------
##   seqinfo: 16 sequences from an unspecified genome

table(seqnames(regions(hic)))
##
##     I   II  III   IV    V   VI  VII VIII   IX    X   XI  XII XIII  XIV   XV
##     0  407    0    0    0    0    0    0    0    0    0    0    0    0    0
##   XVI
##     0

anchors(hic) # ---- `anchors()` work on `HiCExperiment` the same way than on `GInteractions`
##  $first
##  GRanges object with 34063 ranges and 4 metadata columns:
##             seqnames        ranges strand |    bin_id    weight   chr
##                <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##        [1]       II        1-2000      * |       116       NaN    II
##        [2]       II        1-2000      * |       116       NaN    II
##        [3]       II        1-2000      * |       116       NaN    II
##        [4]       II        1-2000      * |       116       NaN    II
##        [5]       II        1-2000      * |       116       NaN    II
##        ...      ...           ...    ... .       ...       ...   ...
##    [34059]       II 804001-806000      * |       518 0.0493107    II
##    [34060]       II 806001-808000      * |       519 0.0611355    II
##    [34061]       II 806001-808000      * |       519 0.0611355    II
##    [34062]       II 806001-808000      * |       519 0.0611355    II
##    [34063]       II 808001-810000      * |       520       NaN    II
##               center
```

```
##             <integer>
##        [1]      1000
##        [2]      1000
##        [3]      1000
##        [4]      1000
##        [5]      1000
##        ...       ...
##    [34059]    805000
##    [34060]    807000
##    [34061]    807000
##    [34062]    807000
##    [34063]    809000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
##
##  $second
##  GRanges object with 34063 ranges and 4 metadata columns:
##             seqnames        ranges strand |    bin_id    weight   chr
##                <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##        [1]       II        1-2000      * |       116       NaN    II
##        [2]       II     4001-6000      * |       118       NaN    II
##        [3]       II     6001-8000      * |       119       NaN    II
##        [4]       II    8001-10000      * |       120 0.0461112    II
##        [5]       II   10001-12000      * |       121 0.0334807    II
##        ...      ...           ...    ... .       ...       ...   ...
##    [34059]       II 810001-812000      * |       521       NaN    II
##    [34060]       II 806001-808000      * |       519 0.0611355    II
##    [34061]       II 808001-810000      * |       520       NaN    II
##    [34062]       II 810001-812000      * |       521       NaN    II
##    [34063]       II 808001-810000      * |       520       NaN    II
##                center
##             <integer>
##        [1]      1000
##        [2]      5000
##        [3]      7000
##        [4]      9000
##        [5]     11000
##        ...       ...
##    [34059]    811000
```

```
##    [34060]    807000
##    [34061]    809000
##    [34062]    811000
##    [34063]    809000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

- Import interactions within a segment of a chromosome:

```
hic <- import(cf, focus = 'II:40000-60000', resolution = 1000)

regions(hic)
##  GRanges object with 21 ranges and 4 metadata columns:
##                    seqnames      ranges strand |    bin_id    weight   chr
##                       <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##    II_39001_40000       II 39001-40000      * |       270 0.0220798    II
##    II_40001_41000       II 40001-41000      * |       271 0.0246775    II
##    II_41001_42000       II 41001-42000      * |       272 0.0269232    II
##    II_42001_43000       II 42001-43000      * |       273 0.0341849    II
##    II_43001_44000       II 43001-44000      * |       274 0.0265386    II
##               ...      ...          ...    ... .       ...       ...   ...
##    II_55001_56000       II 55001-56000      * |       286 0.0213532    II
##    II_56001_57000       II 56001-57000      * |       287 0.0569839    II
##    II_57001_58000       II 57001-58000      * |       288 0.0338612    II
##    II_58001_59000       II 58001-59000      * |       289 0.0294531    II
##    II_59001_60000       II 59001-60000      * |       290 0.0306662    II
##                      center
##                   <integer>
##    II_39001_40000     39500
##    II_40001_41000     40500
##    II_41001_42000     41500
##    II_42001_43000     42500
##    II_43001_44000     43500
##               ...       ...
##    II_55001_56000     55500
##    II_56001_57000     56500
##    II_57001_58000     57500
##    II_58001_59000     58500
##    II_59001_60000     59500
```

```
##    -------
##    seqinfo: 16 sequences from an unspecified genome

anchors(hic)
##  $first
##  GRanges object with 210 ranges and 4 metadata columns:
##          seqnames      ranges strand |    bin_id    weight   chr    center
##             <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle> <integer>
##      [1]       II 40001-41000      * |       271 0.0246775    II     40500
##      [2]       II 40001-41000      * |       271 0.0246775    II     40500
##      [3]       II 40001-41000      * |       271 0.0246775    II     40500
##      [4]       II 40001-41000      * |       271 0.0246775    II     40500
##      [5]       II 40001-41000      * |       271 0.0246775    II     40500
##      ...      ...         ...    ... .       ...       ...   ...       ...
##    [206]       II 57001-58000      * |       288 0.0338612    II     57500
##    [207]       II 57001-58000      * |       288 0.0338612    II     57500
##    [208]       II 58001-59000      * |       289 0.0294531    II     58500
##    [209]       II 58001-59000      * |       289 0.0294531    II     58500
##    [210]       II 59001-60000      * |       290 0.0306662    II     59500
##    -------
##    seqinfo: 16 sequences from an unspecified genome
##
##  $second
##  GRanges object with 210 ranges and 4 metadata columns:
##          seqnames      ranges strand |    bin_id    weight   chr    center
##             <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle> <integer>
##      [1]       II 40001-41000      * |       271 0.0246775    II     40500
##      [2]       II 41001-42000      * |       272 0.0269232    II     41500
##      [3]       II 42001-43000      * |       273 0.0341849    II     42500
##      [4]       II 43001-44000      * |       274 0.0265386    II     43500
##      [5]       II 44001-45000      * |       275 0.0488968    II     44500
##      ...      ...         ...    ... .       ...       ...   ...       ...
##    [206]       II 58001-59000      * |       289 0.0294531    II     58500
##    [207]       II 59001-60000      * |       290 0.0306662    II     59500
##    [208]       II 58001-59000      * |       289 0.0294531    II     58500
##    [209]       II 59001-60000      * |       290 0.0306662    II     59500
##    [210]       II 59001-60000      * |       290 0.0306662    II     59500
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

- Import interactions between two chromosomes:

```
hic2 <- import(cf, focus = 'II|XV', resolution = 4000)

regions(hic2)
##  GRanges object with 477 ranges and 4 metadata columns:
##                         seqnames          ranges strand |    bin_id    weight
##                            <Rle>       <IRanges>  <Rle> | <numeric> <numeric>
##             II_1_4000       II          1-4000      * |        58       NaN
##          II_4001_8000       II       4001-8000      * |        59       NaN
##         II_8001_12000       II      8001-12000      * |        60 0.0274474
##        II_12001_16000       II     12001-16000      * |        61 0.0342116
##        II_16001_20000       II     16001-20000      * |        62 0.0195128
##                   ...      ...             ...    ... .       ...       ...
##    XV_1072001_1076000       XV 1072001-1076000      * |      2783  0.041763
##    XV_1076001_1080000       XV 1076001-1080000      * |      2784       NaN
##    XV_1080001_1084000       XV 1080001-1084000      * |      2785       NaN
##    XV_1084001_1088000       XV 1084001-1088000      * |      2786       NaN
##    XV_1088001_1091291       XV 1088001-1091291      * |      2787       NaN
##                         chr    center
##                       <Rle> <integer>
##             II_1_4000    II      2000
##          II_4001_8000    II      6000
##         II_8001_12000    II     10000
##        II_12001_16000    II     14000
##        II_16001_20000    II     18000
##                   ...   ...       ...
##    XV_1072001_1076000    XV   1074000
##    XV_1076001_1080000    XV   1078000
##    XV_1080001_1084000    XV   1082000
##    XV_1084001_1088000    XV   1086000
##    XV_1088001_1091291    XV   1089646
##    -------
##    seqinfo: 16 sequences from an unspecified genome

anchors(hic2)
##  $first
##  GRanges object with 18032 ranges and 4 metadata columns:
##             seqnames        ranges strand |    bin_id    weight    chr
##                <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
```

```
##         [1]       II        1-4000      * |        58       NaN    II
##         [2]       II        1-4000      * |        58       NaN    II
##         [3]       II        1-4000      * |        58       NaN    II
##         [4]       II        1-4000      * |        58       NaN    II
##         [5]       II        1-4000      * |        58       NaN    II
##         ...      ...           ...    ... .       ...       ...   ...
##    [18028]       II 808001-812000      * |       260       NaN    II
##    [18029]       II 808001-812000      * |       260       NaN    II
##    [18030]       II 808001-812000      * |       260       NaN    II
##    [18031]       II 808001-812000      * |       260       NaN    II
##    [18032]       II 808001-812000      * |       260       NaN    II
##               center
##            <integer>
##         [1]      2000
##         [2]      2000
##         [3]      2000
##         [4]      2000
##         [5]      2000
##         ...       ...
##    [18028]    810000
##    [18029]    810000
##    [18030]    810000
##    [18031]    810000
##    [18032]    810000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
##
##  $second
##  GRanges object with 18032 ranges and 4 metadata columns:
##             seqnames          ranges strand |    bin_id    weight   chr
##                <Rle>       <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##         [1]       XV     48001-52000      * |      2527 0.0185354    XV
##         [2]       XV   348001-352000      * |      2602 0.0233750    XV
##         [3]       XV   468001-472000      * |      2632 0.0153615    XV
##         [4]       XV   472001-476000      * |      2633 0.0189624    XV
##         [5]       XV   584001-588000      * |      2661 0.0167715    XV
##         ...      ...             ...    ... .       ...       ...   ...
##    [18028]       XV   980001-984000      * |      2760 0.0187827    XV
##    [18029]       XV   984001-988000      * |      2761 0.0250094    XV
##    [18030]       XV   992001-996000      * |      2763 0.0185599    XV
```

```
##    [18031]       XV 1004001-1008000      * |      2766 0.0196942    XV
##    [18032]       XV 1064001-1068000      * |      2781 0.0208220    XV
##               center
##            <integer>
##        [1]     50000
##        [2]    350000
##        [3]    470000
##        [4]    474000
##        [5]    586000
##        ...       ...
##    [18028]    982000
##    [18029]    986000
##    [18030]    994000
##    [18031]   1006000
##    [18032]   1066000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

- Import interactions between segments of two chromosomes:

```
hic3 <- import(cf, focus = 'III:10000-40000|XV:10000-40000', resolution = 2000)

regions(hic3)
##  GRanges object with 32 ranges and 4 metadata columns:
##                     seqnames      ranges strand |    bin_id    weight   chr
##                        <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##     III_8001_10000       III  8001-10000      * |       527       NaN   III
##    III_10001_12000       III 10001-12000      * |       528       NaN   III
##    III_12001_14000       III 12001-14000      * |       529       NaN   III
##    III_14001_16000       III 14001-16000      * |       530 0.0356351   III
##    III_16001_18000       III 16001-18000      * |       531 0.0230693   III
##                ...       ...         ...    ... .       ...       ...   ...
##     XV_30001_32000        XV 30001-32000      * |      5039 0.0482465    XV
##     XV_32001_34000        XV 32001-34000      * |      5040 0.0241580    XV
##     XV_34001_36000        XV 34001-36000      * |      5041 0.0273166    XV
##     XV_36001_38000        XV 36001-38000      * |      5042 0.0542235    XV
##     XV_38001_40000        XV 38001-40000      * |      5043 0.0206849    XV
##                       center
```

```
##                     <integer>
##     III_8001_10000      9000
##    III_10001_12000     11000
##    III_12001_14000     13000
##    III_14001_16000     15000
##    III_16001_18000     17000
##                ...       ...
##     XV_30001_32000     31000
##     XV_32001_34000     33000
##     XV_34001_36000     35000
##     XV_36001_38000     37000
##     XV_38001_40000     39000
##    -------
##    seqinfo: 16 sequences from an unspecified genome

anchors(hic3)
##  $first
##  GRanges object with 11 ranges and 4 metadata columns:
##         seqnames      ranges strand |    bin_id    weight   chr    center
##            <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle> <integer>
##     [1]      III 14001-16000      * |       530 0.0356351   III     15000
##     [2]      III 16001-18000      * |       531 0.0230693   III     17000
##     [3]      III 16001-18000      * |       531 0.0230693   III     17000
##     [4]      III 20001-22000      * |       533 0.0343250   III     21000
##     [5]      III 22001-24000      * |       534 0.0258604   III     23000
##     [6]      III 24001-26000      * |       535 0.0290757   III     25000
##     [7]      III 28001-30000      * |       537 0.0290713   III     29000
##     [8]      III 30001-32000      * |       538 0.0266373   III     31000
##     [9]      III 32001-34000      * |       539 0.0201137   III     33000
##    [10]      III 32001-34000      * |       539 0.0201137   III     33000
##    [11]      III 36001-38000      * |       541 0.0220603   III     37000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
##
##  $second
##  GRanges object with 11 ranges and 4 metadata columns:
##         seqnames      ranges strand |    bin_id    weight   chr    center
##            <Rle>   <IRanges>  <Rle> | <numeric> <numeric> <Rle> <integer>
##     [1]       XV 16001-18000      * |      5032 0.0187250    XV     17000
##     [2]       XV 16001-18000      * |      5032 0.0187250    XV     17000
```

```
##     [3]       XV 20001-22000      * |      5034 0.0247973    XV     21000
##     [4]       XV 14001-16000      * |      5031 0.0379727    XV     15000
##     [5]       XV 10001-12000      * |      5029 0.0296913    XV     11000
##     [6]       XV 32001-34000      * |      5040 0.0241580    XV     33000
##     [7]       XV 16001-18000      * |      5032 0.0187250    XV     17000
##     [8]       XV 38001-40000      * |      5043 0.0206849    XV     39000
##     [9]       XV 22001-24000      * |      5035 0.0613856    XV     23000
##    [10]       XV 30001-32000      * |      5039 0.0482465    XV     31000
##    [11]       XV 10001-12000      * |      5029 0.0296913    XV     11000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

2.4.2 Interacting with `HiCExperiment` data

- An `HiCExperiment` object allows parsing of a disk-stored contact matrix.
- An `HiCExperiment` object operates by wrapping together (1) a `ContactFile` (i.e. a connection to a disk-stored data file) and (2) a `GInteractions` generated by parsing the data file.

We will use the `yeast_hic HiCExperiment` object to demonstrate how to parse information from a `HiCExperiment` object.

```
yeast_hic <- contacts_yeast()
```

```
yeast_hic
##  `HiCExperiment` object with 8,757,906 contacts over 763 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "whole genome"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 16000
##  interactions: 267709
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: /root/.cache/R/ExperimentHub/174ff8a7b2_7753
```

```
##  metadata(3): ID org date
```

2.4.2.1 Interactions

The imported genomic interactions can be directly **exposed** using the `interactions` function and are returned as a `GInteractions` object.

```
interactions(yeast_hic)
##  GInteractions object with 267709 interactions and 4 metadata columns:
##              seqnames1       ranges1     seqnames2       ranges2 |   bin_id1
##                  <Rle>     <IRanges>         <Rle>     <IRanges> | <numeric>
##          [1]         I       1-16000 ---         I       1-16000 |         0
##          [2]         I       1-16000 ---         I   16001-32000 |         0
##          [3]         I       1-16000 ---         I   32001-48000 |         0
##          [4]         I       1-16000 ---         I   48001-64000 |         0
##          [5]         I       1-16000 ---         I   64001-80000 |         0
##          ...       ...           ... ...       ...           ... .       ...
##     [267705]       XVI 896001-912000 ---       XVI 912001-928000 |       759
##     [267706]       XVI 896001-912000 ---       XVI 928001-944000 |       759
##     [267707]       XVI 912001-928000 ---       XVI 912001-928000 |       760
##     [267708]       XVI 912001-928000 ---       XVI 928001-944000 |       760
##     [267709]       XVI 928001-944000 ---       XVI 928001-944000 |       761
##                bin_id2     count  balanced
##              <numeric> <numeric> <numeric>
##          [1]         0      2836 1.0943959
##          [2]         1      2212 0.9592069
##          [3]         2      1183 0.4385242
##          [4]         3       831 0.2231192
##          [5]         4       310 0.0821255
##          ...       ...       ...       ...
##     [267705]       760      3565  1.236371
##     [267706]       761      1359  0.385016
##     [267707]       760      3534  2.103988
##     [267708]       761      3055  1.485794
##     [267709]       761      4308  1.711565
##     -------
##     regions: 763 ranges and 4 metadata columns
```

```
##    seqinfo: 16 sequences from an unspecified genome
```

Because genomic interactions are actually stored as `GInteractions`, `regions` and `anchors` work on `HiCExperiment` objects just as they work with `GInteractions`!

```
regions(yeast_hic)
##  GRanges object with 763 ranges and 4 metadata columns:
##                       seqnames        ranges strand |    bin_id     weight
##                          <Rle>     <IRanges>  <Rle> | <numeric>  <numeric>
##            I_1_16000         I       1-16000      * |         0  0.0196442
##        I_16001_32000         I   16001-32000      * |         1  0.0220746
##        I_32001_48000         I   32001-48000      * |         2  0.0188701
##        I_48001_64000         I   48001-64000      * |         3  0.0136679
##        I_64001_80000         I   64001-80000      * |         4  0.0134860
##                  ...       ...           ...    ... .       ...        ...
##    XVI_880001_896000       XVI 880001-896000      * |       758 0.00910873
##    XVI_896001_912000       XVI 896001-912000      * |       759 0.01421350
##    XVI_912001_928000       XVI 912001-928000      * |       760 0.02439992
##    XVI_928001_944000       XVI 928001-944000      * |       761 0.01993237
##    XVI_944001_948066       XVI 944001-948066      * |       762        NaN
##                         chr    center
##                        <Rle> <integer>
##            I_1_16000     I      8000
##        I_16001_32000     I     24000
##        I_32001_48000     I     40000
##        I_48001_64000     I     56000
##        I_64001_80000     I     72000
##                  ...   ...       ...
##    XVI_880001_896000   XVI    888000
##    XVI_896001_912000   XVI    904000
##    XVI_912001_928000   XVI    920000
##    XVI_928001_944000   XVI    936000
##    XVI_944001_948066   XVI    946033
##    -------
##    seqinfo: 16 sequences from an unspecified genome

anchors(yeast_hic)
##  $first
```

```
##  GRanges object with 267709 ranges and 4 metadata columns:
##              seqnames        ranges strand |    bin_id    weight   chr
##                 <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##         [1]        I       1-16000      * |         0 0.0196442     I
##         [2]        I       1-16000      * |         0 0.0196442     I
##         [3]        I       1-16000      * |         0 0.0196442     I
##         [4]        I       1-16000      * |         0 0.0196442     I
##         [5]        I       1-16000      * |         0 0.0196442     I
##         ...      ...           ...    ... .       ...       ...   ...
##    [267705]      XVI 896001-912000      * |       759 0.0142135   XVI
##    [267706]      XVI 896001-912000      * |       759 0.0142135   XVI
##    [267707]      XVI 912001-928000      * |       760 0.0243999   XVI
##    [267708]      XVI 912001-928000      * |       760 0.0243999   XVI
##    [267709]      XVI 928001-944000      * |       761 0.0199324   XVI
##                center
##             <integer>
##         [1]      8000
##         [2]      8000
##         [3]      8000
##         [4]      8000
##         [5]      8000
##         ...       ...
##    [267705]    904000
##    [267706]    904000
##    [267707]    920000
##    [267708]    920000
##    [267709]    936000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
##
##  $second
##  GRanges object with 267709 ranges and 4 metadata columns:
##              seqnames        ranges strand |    bin_id    weight   chr
##                 <Rle>     <IRanges>  <Rle> | <numeric> <numeric> <Rle>
##         [1]        I       1-16000      * |         0 0.0196442     I
##         [2]        I   16001-32000      * |         1 0.0220746     I
##         [3]        I   32001-48000      * |         2 0.0188701     I
##         [4]        I   48001-64000      * |         3 0.0136679     I
##         [5]        I   64001-80000      * |         4 0.0134860     I
##         ...      ...           ...    ... .       ...       ...   ...
```

```
##    [267705]      XVI 912001-928000      * |       760 0.0243999   XVI
##    [267706]      XVI 928001-944000      * |       761 0.0199324   XVI
##    [267707]      XVI 912001-928000      * |       760 0.0243999   XVI
##    [267708]      XVI 928001-944000      * |       761 0.0199324   XVI
##    [267709]      XVI 928001-944000      * |       761 0.0199324   XVI
##                center
##             <integer>
##         [1]      8000
##         [2]     24000
##         [3]     40000
##         [4]     56000
##         [5]     72000
##         ...       ...
##    [267705]    920000
##    [267706]    936000
##    [267707]    920000
##    [267708]    936000
##    [267709]    936000
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

2.4.2.2 Bins and seqinfo

Additional useful information can be recovered from a `HiCExperiment` object. This includes:

- The `seqinfo` of the `HiCExperiment`:

```
seqinfo(yeast_hic)
##  Seqinfo object with 16 sequences from an unspecified genome:
##    seqnames seqlengths isCircular genome
##    I            230218       <NA>   <NA>
##    II           813184       <NA>   <NA>
##    III          316620       <NA>   <NA>
##    IV          1531933       <NA>   <NA>
##    V            576874       <NA>   <NA>
##    ...             ...        ...    ...
##    XII         1078177       <NA>   <NA>
##    XIII         924431       <NA>   <NA>
```

```
##    XIV          784333       <NA>   <NA>
##    XV          1091291       <NA>   <NA>
##    XVI          948066       <NA>   <NA>
```

This lists the different chromosomes available to parse along with their length.

- The `bins` of the `HiCExperiment`:

```
bins(yeast_hic)
##  GRanges object with 763 ranges and 2 metadata columns:
##                       seqnames        ranges strand |    bin_id     weight
##                          <Rle>     <IRanges>  <Rle> | <numeric>  <numeric>
##            I_1_16000         I       1-16000      * |         0  0.0196442
##        I_16001_32000         I   16001-32000      * |         1  0.0220746
##        I_32001_48000         I   32001-48000      * |         2  0.0188701
##        I_48001_64000         I   48001-64000      * |         3  0.0136679
##        I_64001_80000         I   64001-80000      * |         4  0.0134860
##                  ...       ...           ...    ... .       ...        ...
##    XVI_880001_896000       XVI 880001-896000      * |       758 0.00910873
##    XVI_896001_912000       XVI 896001-912000      * |       759 0.01421350
##    XVI_912001_928000       XVI 912001-928000      * |       760 0.02439992
##    XVI_928001_944000       XVI 928001-944000      * |       761 0.01993237
##    XVI_944001_948066       XVI 944001-948066      * |       762        NaN
##    -------
##    seqinfo: 16 sequences from an unspecified genome
```

> ⚠ Difference between `bins` and `regions`
>
> `bins` are **not** equivalent to `regions` of an `HiCExperiment`.
>
> - `bins` refer to all the **possible** `regions` of a `HiCExperiment`. For instance, for a `HiCExperiment` with a total genome size of `1,000,000` and a resolution of `2000`, `bins` will always return a `GRanges` object with `500` ranges.
> - `regions`, on the opposite, refer to the union of `anchors` of all the `interactions` imported in a `HiCExperiment` object.
>
> Thus, all the `regions` will necessarily be a subset of the

`HiCExperiment` `bins`, or equal to `bins` if no focus has been specified when importing a `ContactFile`.

2.4.2.3 Scores

Of course, what the end-user would be looking for is the **frequency** for each genomic interaction. Such frequency scores are available using the `scores` function. `scores` returns a list with a number of different types of scores.

```
head(scores(yeast_hic))
##  List of length 2
##  names(2): count balanced

head(scores(yeast_hic, "count"))
##  [1] 2836 2212 1183  831  310  159

head(scores(yeast_hic, "balanced"))
##  [1] 1.09439586 0.95920688 0.43852417 0.22311917 0.08212549 0.03345221
```

Calling `interactions(hic)` returns a `GInteractions` with `scores` already stored in extra columns. This short-hand allows one to dynamically check `scores` directly from the `interactions` output.

```
interactions(yeast_hic)
##  GInteractions object with 267709 interactions and 4 metadata columns:
##              seqnames1       ranges1     seqnames2       ranges2 |   bin_id1
##                  <Rle>     <IRanges>         <Rle>     <IRanges> | <numeric>
##          [1]         I       1-16000 ---         I       1-16000 |         0
##          [2]         I       1-16000 ---         I   16001-32000 |         0
##          [3]         I       1-16000 ---         I   32001-48000 |         0
##          [4]         I       1-16000 ---         I   48001-64000 |         0
##          [5]         I       1-16000 ---         I   64001-80000 |         0
##          ...       ...           ... ...       ...           ... .       ...
##     [267705]       XVI 896001-912000 ---       XVI 912001-928000 |       759
##     [267706]       XVI 896001-912000 ---       XVI 928001-944000 |       759
##     [267707]       XVI 912001-928000 ---       XVI 912001-928000 |       760
```

```
##     [267708]       XVI 912001-928000 ---       XVI 928001-944000 |       760
##     [267709]       XVI 928001-944000 ---       XVI 928001-944000 |       761
##                 bin_id2     count  balanced
##               <numeric> <numeric> <numeric>
##           [1]         0      2836 1.0943959
##           [2]         1      2212 0.9592069
##           [3]         2      1183 0.4385242
##           [4]         3       831 0.2231192
##           [5]         4       310 0.0821255
##           ...       ...       ...       ...
##     [267705]       760      3565  1.236371
##     [267706]       761      1359  0.385016
##     [267707]       760      3534  2.103988
##     [267708]       761      3055  1.485794
##     [267709]       761      4308  1.711565
##     -------
##     regions: 763 ranges and 4 metadata columns
##     seqinfo: 16 sequences from an unspecified genome

head(interactions(yeast_hic)$count)
##  [1] 2836 2212 1183  831  310  159
```

2.4.2.4 topologicalFeatures

In Hi-C studies, "topological features" refer to genomic structures identified (usually from a Hi-C map, but not necessarily). For instance, one may want to study known structural loops anchored at CTCF sites, or interactions around or over centromeres, or simply specific genomic "viewpoints".

`HiCExperiment` objects can store `topologicalFeatures` to facilitate this analysis. By default, four empty `topologicalFeatures` are stored in a list:

- `compartments`
- `borders`
- `loops`
- `viewpoints`

Additional `topologicalFeatures` can be added to this list (read next chapter for more detail).

```
topologicalFeatures(yeast_hic)
##  List of length 5
##  names(5): compartments borders loops viewpoints centromeres

topologicalFeatures(yeast_hic, 'centromeres')
##  GRanges object with 16 ranges and 0 metadata columns:
##         seqnames        ranges strand
##            <Rle>     <IRanges>  <Rle>
##     [1]        I 151583-151641      +
##     [2]       II 238361-238419      +
##     [3]      III 114322-114380      +
##     [4]       IV 449879-449937      +
##     [5]        V 152522-152580      +
##     ...      ...           ...    ...
##    [12]      XII 151366-151424      +
##    [13]     XIII 268222-268280      +
##    [14]      XIV 628588-628646      +
##    [15]       XV 326897-326955      +
##    [16]      XVI 556255-556313      +
##    -------
##    seqinfo: 17 sequences (1 circular) from R64-1-1 genome
```

2.4.2.5 pairsFile

As a contact matrix is typically obtained from binning a `.pairs` file, it is often the case that the matching `.pairs` file is available to then end-user. A `PairsFile` can thus be created and associated to the corresponding `HiCExperiment` object. This allows more accurate estimation of contact distribution, e.g. when calculating distance-dependent genomic interaction frequency.

```
pairsFile(yeast_hic) <- pairsf

pairsFile(yeast_hic)
##                                                EH7703
##  "/root/.cache/R/ExperimentHub/174ff8a7b2_7753"
```

```
readLines(pairsFile(yeast_hic), 25)
##   [1] "## pairs format v1.0"
##   [2] "#sorted: chr1-pos1-chr2-pos2"
##   [3] "#columns: readID chr1 pos1 chr2 pos2 strand1 strand2 frag1 frag2"
##   [4] "#chromsize: I 230218"
##   [5] "#chromsize: II 813184"
##   [6] "#chromsize: III 316620"
##   [7] "#chromsize: IV 1531933"
##   [8] "#chromsize: V 576874"
##   [9] "#chromsize: VI 270161"
##  [10] "#chromsize: VII 1090940"
##  [11] "#chromsize: VIII 562643"
##  [12] "#chromsize: IX 439888"
##  [13] "#chromsize: X 745751"
##  [14] "#chromsize: XI 666816"
##  [15] "#chromsize: XII 1078177"
##  [16] "#chromsize: XIII 924431"
##  [17] "#chromsize: XIV 784333"
##  [18] "#chromsize: XV 1091291"
##  [19] "#chromsize: XVI 948066"
##  [20] "#chromsize: Mito 85779"
##  [21] "NS500150:527:HHGYNBGXF:3:21611:19085:3986\tII\t105\tII\t48548\t+\t-\t1358\t1681"
##  [22] "NS500150:527:HHGYNBGXF:4:13604:19734:2406\tII\t113\tII\t45003\t-\t+\t1358\t1658"
##  [23] "NS500150:527:HHGYNBGXF:2:11108:25178:11036\tII\t119\tII\t687251\t-\t+\t1358\t5550"
##  [24] "NS500150:527:HHGYNBGXF:1:22301:8468:1586\tII\t160\tII\t26124\t+\t-\t1358\t1510"
##  [25] "NS500150:527:HHGYNBGXF:4:23606:24037:2076\tII\t169\tII\t39052\t+\t+\t1358\t1613"
```

2.4.2.6 Importing a `PairsFile`

The `.pairs` file linked to a `HiCExperiment` object can itself be imported in a `GInteractions` object:

```
import(pairsFile(yeast_hic), format = 'pairs')
##  GInteractions object with 471364 interactions and 3 metadata columns:
##              seqnames1   ranges1     seqnames2   ranges2 |     frag1     frag2
##                  <Rle> <IRanges>         <Rle> <IRanges> | <numeric> <numeric>
##          [1]        II       105 ---        II     48548 |      1358      1681
##          [2]        II       113 ---        II     45003 |      1358      1658
```

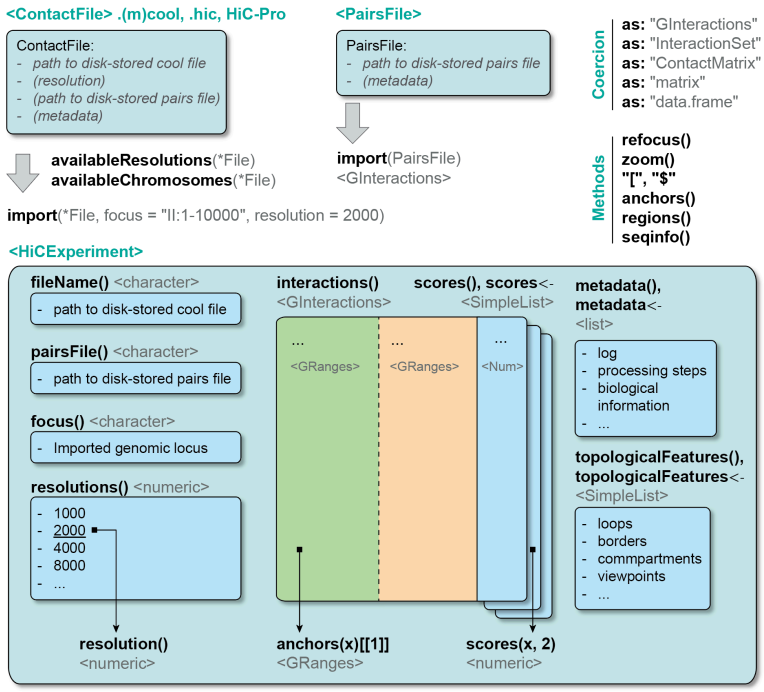
```
##         [3]        II       119 ---        II    687251 |      1358      5550
##         [4]        II       160 ---        II     26124 |      1358      1510
##         [5]        II       169 ---        II     39052 |      1358      1613
##         ...       ...       ... ...       ...       ... .       ...       ...
##    [471360]        II    808605 ---        II    809683 |      6316      6320
##    [471361]        II    808609 ---        II    809917 |      6316      6324
##    [471362]        II    808617 ---        II    809506 |      6316      6319
##    [471363]        II    809447 ---        II    809685 |      6319      6321
##    [471364]        II    809472 ---        II    809675 |      6319      6320
##                distance
##               <integer>
##         [1]     48443
##         [2]     44890
##         [3]    687132
##         [4]     25964
##         [5]     38883
##         ...       ...
##    [471360]      1078
##    [471361]      1308
##    [471362]       889
##    [471363]       238
##    [471364]       203
##    -------
##    regions: 549331 ranges and 0 metadata columns
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Note that these `GInteractions` are **not** binned, contrary to `interactions` extracted from a `HiCExperiment`. Anchors of the interactions listed in the `GInteractions` imported from a disk-stored `.pairs` file are all of width `1`.

2.5 Visual summary of the `HiCExperiment` data structure

The `HiCExperiment` data structure provided by the `HiCExperiment` package inherits methods from core `GInteractions` and `BiocFile` classes to provide a flexible representation of Hi-C data in `R`. It allows random access-based queries to seamlessly import parts or all the data

contained in disk-stored Hi-C contact matrices in a variety of formats.



References

3 Manipulating Hi-C data in R

Aims

This chapter focuses on:

- Modifying information associated with an existing `HiCExperiment` object
- Subsetting a `HiCExperiment` object
- Coercing a `HiCExperiment` object in a base data structure

Important reminder

- An `HiCExperiment` object allows random access parsing of a disk-stored contact matrix.
- An `HiCExperiment` object operates by wrapping together (1) a `ContactFile` (i.e. a connection to a disk-stored data file) and (2) a `GInteractions` generated by parsing the data file.

Recap on `HiCExperiment` objects

- Creating a connection to a disk-stored contact matrix:

```
# coolf <- "<path-to-disk-stored-contact-matrix.cool>"
coolf <- HiContactsData('yeast_wt', 'mcool')
cf <- CoolFile(coolf)
availableResolutions(cf)

availableChromosomes(cf)
```

- Importing a contact matrix over a specific genomic

location, at a given resolution:

```
hic <- import(cf, focus = 'II:10000-50000', resolution = 4000)
hic
##  `HiCExperiment` object with 10,801 contacts over 11 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:10,000-50,000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 4000
##  interactions: 45
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Recovering genomic interactions stored in a `HiCExperiment`:

```
interactions(hic)
##  GInteractions object with 45 interactions and 4 metadata columns:
##         seqnames1     ranges1     seqnames2     ranges2 |   bin_id1   bin_id2
##             <Rle>   <IRanges>         <Rle>   <IRanges> | <numeric> <numeric>
##     [1]        II 12001-16000 ---        II 12001-16000 |        61        61
##     [2]        II 12001-16000 ---        II 16001-20000 |        61        62
##     [3]        II 12001-16000 ---        II 20001-24000 |        61        63
##     [4]        II 12001-16000 ---        II 24001-28000 |        61        64
##     [5]        II 12001-16000 ---        II 28001-32000 |        61        65
##     ...       ...         ... ...       ...         ... .       ...       ...
##    [41]        II 36001-40000 ---        II 40001-44000 |        67        68
##    [42]        II 36001-40000 ---        II 44001-48000 |        67        69
##    [43]        II 40001-44000 ---        II 40001-44000 |        68        68
##    [44]        II 40001-44000 ---        II 44001-48000 |        68        69
##    [45]        II 44001-48000 ---        II 44001-48000 |        69        69
##             count  balanced
##         <numeric> <numeric>
##     [1]       213  0.249303
##     [2]       673  0.449271
##     [3]       325  0.210001
##     [4]       137  0.125732
##     [5]        77  0.106917
##     ...       ...       ...
##    [41]       941  0.358860
##    [42]       275  0.114972
##    [43]       675  0.253868
##    [44]       497  0.204920
##    [45]       295  0.133344
##    -------
##    regions: 11 ranges and 4 metadata columns
##    seqinfo: 16 sequences from an unspecified genome
```

Generating the example `hic` object

To demonstrate how to manipulate a `HiCExperiment` object, we will create an `HiCExperiment` object from an example `.cool` file provided in the `HiContactsData` package.

```
library(HiCExperiment)
library(HiContactsData)

# ---- This downloads an example `.mcool` file and caches it locally
coolf <- HiContactsData('yeast_wt', 'mcool')
##  see ?HiContactsData and browseVignettes('HiContactsData') for documentation
##  loading from cache

# ---- This creates a connection to the disk-stored `.mcool` file
cf <- CoolFile(coolf)
cf
##  CoolFile object
##  .mcool file: /root/.cache/R/ExperimentHub/174688ce76a_7752
##  resolution: 1000
##  pairs file:
##  metadata(0):

# ---- This imports contacts from the long arm of chromosome `II`, at resolution `2000`
hic <- import(cf, focus = 'II:300001-813184', resolution = 2000)
hic
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

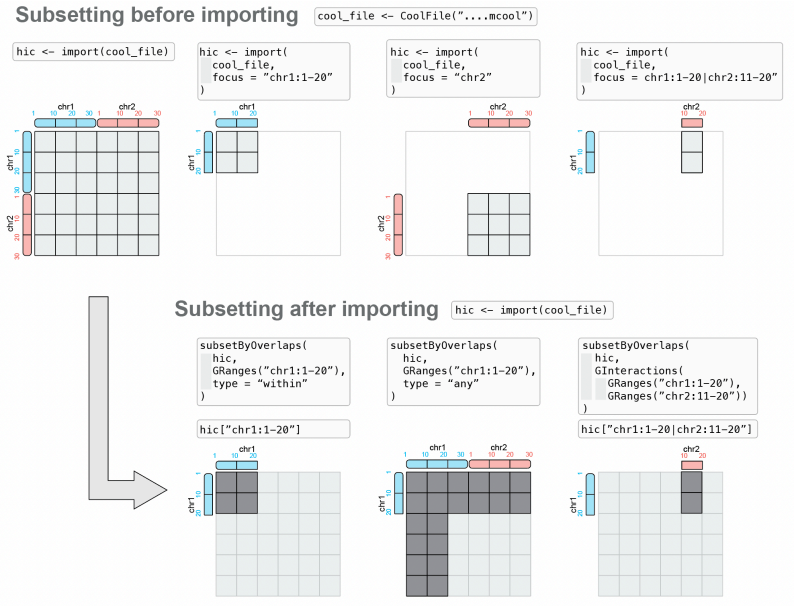
3.1 Subsetting a contact matrix

Two entirely different approaches are possible to subset of a Hi-C contact matrix:

- **Subsetting before importing**: leveraging random ac-

cess to a **disk-stored** contact matrix to **only import interactions overlapping with a genomic locus of interest**.

- **Subsetting after importing**: parsing the **entire contact matrix** in memory, and **subsequently** subset interactions overlapping with a genomic locus of interest.



3.1.1 Subsetting before import: with `focus`

Specifying a `focus` **when** importing a dataset in R (i.e. `"Subset first, then parse"`) is generally the recommended approach to import Hi-C data in R.

The `focus` argument can be set when `import`ing a `ContactFile` in R, as follows:

```
import(cf, focus = "...")
```

This ensures that only the needed data is parsed in R, reducing memory load and accelerating the import. Thus, this should

be the preferred way of parsing `HiCExperiment` data, as disk-stored contact matrices allow **efficient random access to indexed data**.

`focus` can be any of the following string types:

```
#   "II"                                  --> import contacts over an entire chromosome
#   "II:300001-800000"                    --> import on-diagonal contacts within a chromosom
#   "II:300001-400000|II:600001-700000"   --> import off-diagonal contacts within a chromoso
#   "II|III"                              --> import contacts between two chromosomes
#   "II:300001-800000|V:1-500000"         --> import contacts between segments of two chromo
```

! More examples for import with `focus` argument

- Subsetting to a specific **on-diagonal** genomic location using standard UCSC coordinates query:

```
import(cf, focus = 'II:300001-800000', resolution = 2000)
##  `HiCExperiment` object with 301,018 contacts over 250 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-800,000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 17974
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting to a specific **off-diagonal** genomic location using pairs of coordinates query:

```
import(cf, focus = 'II:300001-400000|II:600001-700000', resolution = 2000)
##  `HiCExperiment` object with 402 contacts over 100 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300001-400000|II:600001-700000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 357
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those constrained within a single chromosome:

```
import(cf, focus = 'II', resolution = 2000)
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 34063
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those between two chromosomes:

```
import(cf, focus = 'II|III', resolution = 2000)
##  `HiCExperiment` object with 9,092 contacts over 566 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II|III"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 7438
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those between parts of two chromosomes:

```
import(cf, focus = 'II:300001-800000|V:1-500000', resolution = 2000)
##  `HiCExperiment` object with 7,147 contacts over 500 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300001-800000|V:1-500000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 6523
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

3.1.2 Subsetting after import

It may sometimes be desirable to import a full dataset from disk first, and **only then** perform in-memory subsetting of the `HiCExperiment` object (i.e. `"Parse first, then subset"`). **This is for example necessary when the end user aims to investigate subsets of interactions across a large number of different areas of a contact matrix.**

Several strategies are possible to allow subsetting of imported data, either with `subsetByOverlaps` or `[`.

3.1.2.1 `subsetByOverlaps(<HiCExperiment>, <GRanges>)`

`subsetByOverlaps` can take a `HiCExperiment` as a `query` and a `GRanges` as a query. In this case, the `GRanges` is used to extract a subset of a `HiCExperiment` **constrained** within a specific genomic location.

```
telomere <- GRanges("II:700001-813184")
subsetByOverlaps(hic, telomere) |> interactions()
##  GInteractions object with 1540 interactions and 4 metadata columns:
##           seqnames1       ranges1     seqnames2       ranges2 |   bin_id1
##               <Rle>     <IRanges>         <Rle>     <IRanges> | <numeric>
##       [1]        II 700001-702000 ---        II 700001-702000 |       466
##       [2]        II 700001-702000 ---        II 702001-704000 |       466
##       [3]        II 700001-702000 ---        II 704001-706000 |       466
##       [4]        II 700001-702000 ---        II 706001-708000 |       466
##       [5]        II 700001-702000 ---        II 708001-710000 |       466
##       ...       ...           ... ...       ...           ... .       ...
##    [1536]        II 804001-806000 ---        II 810001-812000 |       518
##    [1537]        II 806001-808000 ---        II 806001-808000 |       519
##    [1538]        II 806001-808000 ---        II 808001-810000 |       519
##    [1539]        II 806001-808000 ---        II 810001-812000 |       519
##    [1540]        II 808001-810000 ---        II 808001-810000 |       520
##             bin_id2     count  balanced
##           <numeric> <numeric> <numeric>
##       [1]       466        30 0.0283618
##       [2]       467       145 0.0709380
##       [3]       468       124 0.0704979
##       [4]       469        59 0.0510221
##       [5]       470        59 0.0384004
##       ...       ...       ...       ...
##    [1536]       521         1       NaN
##    [1537]       519        15 0.0560633
##    [1538]       520        25       NaN
##    [1539]       521         1       NaN
##    [1540]       520        10       NaN
```

```
##    -------
##    regions: 57 ranges and 4 metadata columns
##    seqinfo: 16 sequences from an unspecified genome
```

By default, `subsetByOverlaps(hic, telomere)` will only recover interactions **constrained** within `telomere`, i.e. interactions for which both ends are in `telomere`.

Alternatively, `type = "any"` can be specified to get all interactions with at least one of their anchors within `telomere`.

```
subsetByOverlaps(hic, telomere, type = "any") |> interactions()
##  GInteractions object with 6041 interactions and 4 metadata columns:
##           seqnames1       ranges1     seqnames2       ranges2 |   bin_id1
##               <Rle>     <IRanges>         <Rle>     <IRanges> | <numeric>
##       [1]        II 300001-302000 ---        II 702001-704000 |       266
##       [2]        II 300001-302000 ---        II 704001-706000 |       266
##       [3]        II 300001-302000 ---        II 768001-770000 |       266
##       [4]        II 300001-302000 ---        II 784001-786000 |       266
##       [5]        II 302001-304000 ---        II 740001-742000 |       267
##       ...       ...           ... ...       ...           ... .       ...
##    [6037]        II 804001-806000 ---        II 810001-812000 |       518
##    [6038]        II 806001-808000 ---        II 806001-808000 |       519
##    [6039]        II 806001-808000 ---        II 808001-810000 |       519
##    [6040]        II 806001-808000 ---        II 810001-812000 |       519
##    [6041]        II 808001-810000 ---        II 808001-810000 |       520
##             bin_id2     count    balanced
##           <numeric> <numeric>   <numeric>
##       [1]       467         1 0.000590999
##       [2]       468         1 0.000686799
##       [3]       500         1 0.000728215
##       [4]       508         1 0.000923092
##       [5]       486         1 0.000382222
##       ...       ...       ...         ...
##    [6037]       521         1         NaN
##    [6038]       519        15   0.0560633
##    [6039]       520        25         NaN
##    [6040]       521         1         NaN
##    [6041]       520        10         NaN
##    -------
```

```
##    regions: 257 ranges and 4 metadata columns
##    seqinfo: 16 sequences from an unspecified genome
```

3.1.2.2 `<HiCExperiment>["..."]`

The square bracket operator `[` allows for more advanced textual queries, similarly to `focus` arguments that can be used when importing contact matrices in memory.

This ensures that only the needed data is parsed in R, reducing memory load and accelerating the import. Thus, this should be the preferred way of parsing `HiCExperiment` data, as disk-stored contact matrices allow efficient random access to indexed data.

The following string types can be used to subset a `HiCExperiment` object with the `[` notation:

```
#   "II"                                  --> import contacts over an entire chromosome
#   "II:300001-800000"                    --> import on-diagonal contacts within a chromosom
#   "II:300001-400000|II:600001-700000"   --> import off-diagonal contacts within a chromoso
#   "II|III"                              --> import contacts between two chromosomes
#   "II:300001-800000|V:1-500000"         --> import contacts between segments of two chromo
#   c("II", "III", "IV")                  --> import contacts within and between several chr
```

> **!** More examples for subsetting with `[`
>
> - Subsetting to a specific **on-diagonal** genomic location using standard UCSC coordinates query:

```
hic["II:800001-813184"]
##  `HiCExperiment` object with 1,040 contacts over 6 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:800,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 19
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting to a specific **off-diagonal** genomic location using pairs of coordinates query:

```
hic["II:300001-320000|II:800001-813184"]
##  `HiCExperiment` object with 3 contacts over 6 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300001-320000|II:800001-813184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 3
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those constrained within a single chromosome:

```
hic["II"]
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those between two chromosomes:

```
hic["II|IV"]
##  `HiCExperiment` object with 0 contacts over 0 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:1-813184|IV:1-1531933"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 0
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those between segments of two chromosomes:

```
hic["II:300001-320000|IV:1-100000"]
##  `HiCExperiment` object with 0 contacts over 0 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300001-320000|IV:1-100000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 0
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

- Subsetting interactions to retain those constrained within several chromosomes:

```
hic[c('II', 'III', 'IV')]
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II, III, IV"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

Some notes:

- This last example (subsetting for a vector of several chromosomes) is the only scenario for which `[`-based in-memory subsetting of pre-imported data is the only way to go, as such subsetting is not possible with `focus` from disk-stored data.
- All the other `[` subsetting scenarii illustrated above can be achieved more efficiently using the `focus` argument when `import`ing data into a `HiCExperiment` object.

- However, keep in mind that subsetting preserves extra data, e.g. added `scores`, `topologicalFeatures`, `metadata` or `pairsFile`, whereas this information is lost using `focus` with `import`.

3.1.3 Zooming on a `HiCExperiment`

"Zooming" refers to dynamically changing the **resolution** of a `HiCExperiment`. By `zoom`ing a `HiCExperiment`, one can refine or coarsen the contact matrix. This operation takes a`ContactFile` and `focus` from an existing `HiCExperiment` input and re-generates a new `HiCExperiment` with updated `resolution`, `interactions` and `scores`. Note that `zoom` will preserve existing `metadata`, `topologicalFeatures` and `pairsFile` information.

```
hic
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):

zoom(hic, 4000)
##  `HiCExperiment` object with 306,212 contacts over 129 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 4000
##  interactions: 6800
##  scores(2): count balanced
```

```
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):

zoom(hic, 1000)
##  `HiCExperiment` object with 306,212 contacts over 514 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 1000
##  interactions: 44363
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):
```

Note

The sum of raw counts do not change after `zoom`ing, however the number of individual `interactions` and `regions` changes.

```
length(hic)
##  [1] 18513
length(zoom(hic, 1000))
##  [1] 44363
length(zoom(hic, 4000))
##  [1] 6800
sum(scores(hic, "count"))
##  [1] 306212
sum(scores(zoom(hic, 1000), "count"))
##  [1] 306212
sum(scores(zoom(hic, 4000), "count"))
##  [1] 306212
```

> ❗ Important
>
> - `zoom` does not change the `focus`! It only affects the `resolution` (and consequently, the `interactions`).
> - `zoom` will only work for multi-resolution contact matrices, e.g. `.mcool` or `.hic`.

3.2 Updating an `HiCExperiment` object

> 💡 TL;DR: Which `HiCExperiment` slots are mutable ( ) / immutable ( )?
>
> - `fileName(hic)`: (obtained from disk-stored file)
> - `focus(hic)`: (see subsetting section)
> - `resolutions(hic)`: (obtained from disk-stored file)
> - `resolution(hic)`: (see zooming section)
> - `interactions(hic)`: (obtained from disk-stored file)
> - `scores(hic)`:
> - `topologicalFeatures(hic)`:
> - `pairsFile(hic)`:
> - `metadata(hic)`:

3.2.1 Immutable slots

An `HiCExperiment` object acts as an interface exposing disk-stored data. This implies that the `fileName` slot itself is immutable (i.e. **cannot** be changed). This should be obvious, as a `HiCExperiment` *has to* be associated with a disk-stored contact matrix to properly function (except in some advanced cases developed in next chapters).

For this reason, methods to manually modify `interactions` and `resolutions` slots are also **not** exposed in the `HiCExperiment` package.

A corollary of this is that the associated `regions` and `anchors` of an `HiCExperiment` should **not** be modified by hand either, since they are directly linked to `interactions`.

3.2.2 Mutable slots

That being said, `HiCExperiment` objects are flexible and can be partially modified in memory without having to change/overwrite the original, disk-stored contact matrix.

Several `slots` can be modified in memory: `slots`, `topologicalFeatures`, `pairsFile` and `metadata`.

3.2.2.1 `scores`

We have seen in the previous chapter that scores are stored in a `list` and are available using the `scores` function.

```
scores(hic)
##  List of length 2
##  names(2): count balanced

head(scores(hic, "count"))
##  [1]  7 92 75 61 38 43

head(scores(hic, "balanced"))
##  [1] 0.009657438 0.076622340 0.054101992 0.042940512 0.040905212 0.029293930
```

Extra scores can be added to this list, e.g. to describe the "expected" interaction frequency for each interaction stored in the `HiCExperiment` object). This can be achieved using the `scores()<-` function.

```
scores(hic, "random") <- runif(length(hic))

scores(hic)
##  List of length 3
##  names(3): count balanced random
```

```
head(scores(hic, "random"))
##  [1] 0.21574322 0.03400429 0.46656945 0.46060108 0.38438533 0.73706667
```

3.2.2.2 `topologicalFeatures`

The end-user can create additional `topologicalFeatures` or modify the existing ones using the `topologicalFeatures()<-` function.

```
topologicalFeatures(hic, 'CTCF') <- GRanges(c(
    "II:340-352",
    "II:3520-3532",
    "II:7980-7992",
    "II:9240-9252"
))
topologicalFeatures(hic, 'CTCF')
##  GRanges object with 4 ranges and 0 metadata columns:
##        seqnames    ranges strand
##           <Rle> <IRanges>  <Rle>
##    [1]       II   340-352      *
##    [2]       II 3520-3532      *
##    [3]       II 7980-7992      *
##    [4]       II 9240-9252      *
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths

topologicalFeatures(hic, 'loops') <- GInteractions(
    topologicalFeatures(hic, 'CTCF')[rep(1:3, each = 3)],
    topologicalFeatures(hic, 'CTCF')[rep(1:3, 3)]
)
topologicalFeatures(hic, 'loops')
##  GInteractions object with 9 interactions and 0 metadata columns:
##        seqnames1   ranges1     seqnames2   ranges2
##            <Rle> <IRanges>         <Rle> <IRanges>
##    [1]        II   340-352 ---        II   340-352
##    [2]        II   340-352 ---        II 3520-3532
##    [3]        II   340-352 ---        II 7980-7992
##    [4]        II 3520-3532 ---        II   340-352
```

```
##    [5]        II 3520-3532 ---        II 3520-3532
##    [6]        II 3520-3532 ---        II 7980-7992
##    [7]        II 7980-7992 ---        II   340-352
##    [8]        II 7980-7992 ---        II 3520-3532
##    [9]        II 7980-7992 ---        II 7980-7992
##    -------
##    regions: 3 ranges and 0 metadata columns
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths

hic
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(3): count balanced random
##  topologicalFeatures: compartments(0) borders(0) loops(9) viewpoints(0) CTCF(4)
##  pairsFile: N/A
##  metadata(0):
```

All these objects can be used in `*Overlap` methods, as they all extend the `GRanges` class of objects.

```
# ---- This counts the number of times `CTCF` anchors are being used in the
#      `loops` `GInteractions` object
countOverlaps(
    query = topologicalFeatures(hic, 'CTCF'),
    subject = topologicalFeatures(hic, 'loops')
)
##  [1] 5 5 5 0
```

3.2.2.3 `pairsFile`

If `pairsFile` is not specified when importing the `ContactFile` into a `HiCExperiment` object, one can add it later.

```
pairsf <- HiContactsData('yeast_wt', 'pairs.gz')
```

```
pairsFile(hic) <- pairsf
hic
##  `HiCExperiment` object with 306,212 contacts over 257 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:300,001-813,184"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 18513
##  scores(3): count balanced random
##  topologicalFeatures: compartments(0) borders(0) loops(9) viewpoints(0) CTCF(4)
##  pairsFile: /root/.cache/R/ExperimentHub/174ff8a7b2_7753
##  metadata(0):
```

3.2.2.4 `metadata`

Metadata associated with a `HiCExperiment` can be updated at any point.

```
metadata(hic) <- list(
    info = "HiCExperiment created from an example .mcool file from `HiContactsData`",
    date = date()
)
metadata(hic)
##  $info
##  [1] "HiCExperiment created from an example .mcool file from `HiContactsData`"
##
##  $date
##  [1] "Tue Nov  7 18:40:38 2023"
```

3.3 Coercing `HiCExperiment` objects

Convenient coercing functions exist to transform data stored as a `HiCExperiment` into another class.

- `as.matrix()`: allows to coerce the `HiCExperiment` into a `sparse` or `dense` matrix (using the `sparse` logical argument, `TRUE` by default) and choosing specific `scores`

of interest (using the `use.scores` argument, `"balanced"` by default).

```
# ----- `as.matrix` coerces a `HiCExperiment` into a `sparseMatrix` by default
as.matrix(hic) |> class()
##  [1] "dgTMatrix"
##  attr(,"package")
##  [1] "Matrix"

as.matrix(hic) |> dim()
##  [1] 257 257

# ----- One can specify which scores should be used when coercing into a matrix
as.matrix(hic, use.scores = "balanced")[1:5, 1:5]
##  5 x 5 sparse Matrix of class "dgTMatrix"
##
##  [1,] 0.009657438 0.07662234 0.05410199 0.04294051 0.04090521
##  [2,] 0.076622340 0.05128277 0.09841564 0.06926737 0.05263611
##  [3,] 0.054101992 0.09841564 0.05657589 0.08723160 0.07316890
##  [4,] 0.042940512 0.06926737 0.08723160 0.03699543 0.08403496
##  [5,] 0.040905212 0.05263611 0.07316890 0.08403496 0.04787415

as.matrix(hic, use.scores = "count")[1:5, 1:5]
##  5 x 5 sparse Matrix of class "dgTMatrix"
##
##  [1,]  7  92  75  61  38
##  [2,] 92 102 226 163  81
##  [3,] 75 226 150 237 130
##  [4,] 61 163 237 103 153
##  [5,] 38  81 130 153  57

# ----- If **expressly required**, one can coerce a HiCExperiment into a dense matrix
as.matrix(hic, use.scores = "count", sparse = FALSE)[1:5, 1:5]
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    7   92   75   61   38
##  [2,]   92  102  226  163   81
##  [3,]   75  226  150  237  130
##  [4,]   61  163  237  103  153
##  [5,]   38   81  130  153   57
```

- `as.data.frame()`: simply coercing `interactions` into a

rectangular data frame

```
as.data.frame(hic) |> head()
##    seqnames1 start1   end1 width1 strand1 bin_id1    weight1 center1
## 1         II 300001 302000   2000       *     266 0.03714342  301000
## 2         II 300001 302000   2000       *     266 0.03714342  301000
## 3         II 300001 302000   2000       *     266 0.03714342  301000
## 4         II 300001 302000   2000       *     266 0.03714342  301000
## 5         II 300001 302000   2000       *     266 0.03714342  301000
## 6         II 300001 302000   2000       *     266 0.03714342  301000
##    seqnames2 start2   end2 width2 strand2 bin_id2    weight2 center2 count
## 1         II 300001 302000   2000       *     266 0.03714342  301000     7
## 2         II 302001 304000   2000       *     267 0.02242258  303000    92
## 3         II 304001 306000   2000       *     268 0.01942093  305000    75
## 4         II 306001 308000   2000       *     269 0.01895202  307000    61
## 5         II 308001 310000   2000       *     270 0.02898098  309000    38
## 6         II 310001 312000   2000       *     271 0.01834118  311000    43
##       balanced     random
## 1 0.009657438 0.21574322
## 2 0.076622340 0.03400429
## 3 0.054101992 0.46656945
## 4 0.042940512 0.46060108
## 5 0.040905212 0.38438533
## 6 0.029293930 0.73706667
```

> **Warning**
>
> These coercing methods only operate on `interactions` and `scores`, and discard all other information, e.g. regarding genomic `regions`, available `resolutions`, associated `metadata`, `pairsFile` or `topologicalFeatures`.

References

4 Hi-C data visualization

Aims

This chapter focuses on the various visualization tools offered by `HiContacts` to plot `HiCExperiment` contact matrices in R.

Generating the example `hic` object

To demonstrate how to visualize a `HiCExperiment` contact matrix, we will create an `HiCExperiment` object from an example `.cool` file provided in the `HiContactsData` package.

```
library(HiCExperiment)
library(HiContactsData)

# ---- This downloads an example `.mcool` file and caches it locally
coolf <- HiContactsData('yeast_wt', 'mcool')

# ---- This creates a connection to the disk-stored `.mcool` file
cf <- CoolFile(coolf)
cf

# ---- This imports contacts from the chromosome `V` at resolution `2000`
hic <- import(cf, focus = 'V', resolution = 2000)
```

```
hic
##  `HiCExperiment` object with 303,545 contacts over 289 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "V"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 20177
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

4.1 Visualizing Hi-C contact maps

Visualizing Hi-C contact maps is often a necessary step in exploratory data analysis. A Hi-C contact map is usually displayed as a heatmap, in which:

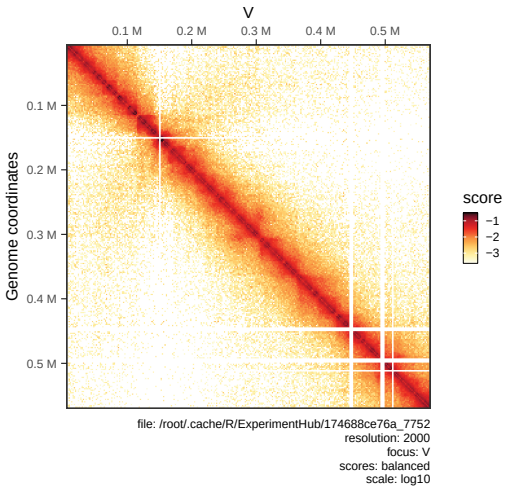
- Each axis represents a section of the genome of interest (either a segment of a chromosome, or several chromosomes, …).
- The color code aims to represent "interaction frequency", which can be expressed in "raw" counts or normalized (balanced).
- Other metrics can also be displayed in Hi-C heatmaps, e.g. ratios of interaction frequency between two Hi-C experiments, p-values of differential interaction analysis, …
- Axes are often identical, representing interactions constrained within a single genomic window, a.k.a. **on-diagonal** matrices.
- However, axes *can* be different: this is the case when **off-diagonal** matrices are displayed.

4.1.1 Single map

Simple visualization of disk-stored Hi-C contact matrices can be done by:

1. Importing the interactions over the genomic location of interest into a `HiCExperiment` object;
2. Using `plotMatrix` function (provided by `HiContacts`) to generate a plot.

```
library(HiContacts)
plotMatrix(hic)
```



> **Note**
>
> A caption summarizing the plotting parameters is added below the heatmap. This can be removed with `caption = FALSE`.

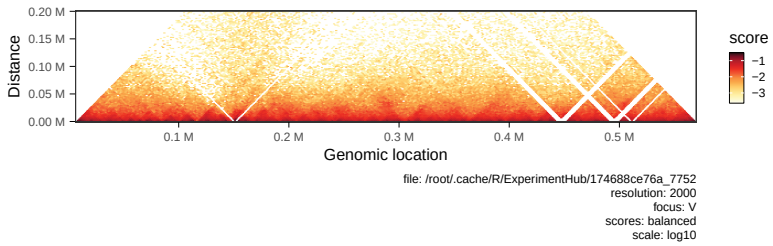
4.1.2 Horizontal map

Hi-C maps are sometimes visualized in a "horizontal" style, where a square **on-diagonal** heatmap is tilted by 45 ° and

truncated to only show interactions up to a certain distance from the main diagonal.

When a `maxDistance` argument is provided to `plotMatrix`, it automatically generates a horizontal-style heatmap.

```
plotMatrix(hic, maxDistance = 200000)
```



4.1.3 Side-by-side maps

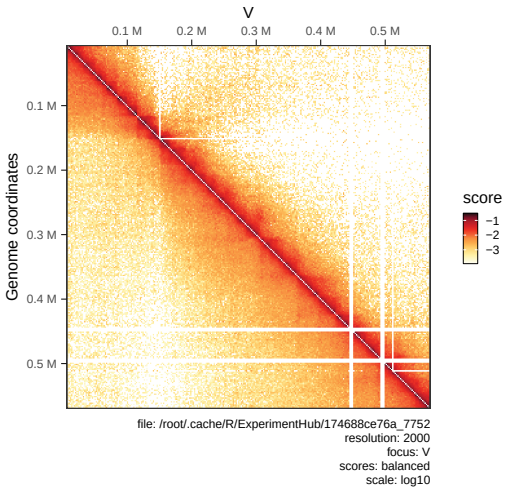
Sometimes, one may want to visually plot 2 Hi-C samples side by side to compare the interaction landscapes over the same genomic locus. This can be done by adding a second `HiCExperiment` (imported with the same `focus`) with the `compare.to` argument.

Here, we are importing a second `.mcool` file corresponding to a Hi-C experiment performed in a *eco1* yeast mutant:

```
hic2 <- import(
    CoolFile(HiContactsData('yeast_eco1', 'mcool')),
    focus = 'V',
    resolution = 2000
)
```

We then plot the 2 matrices side by side. The first will be displayed in the top right corner and the second (provided with `compare.to`) will be in the bottom left corner.

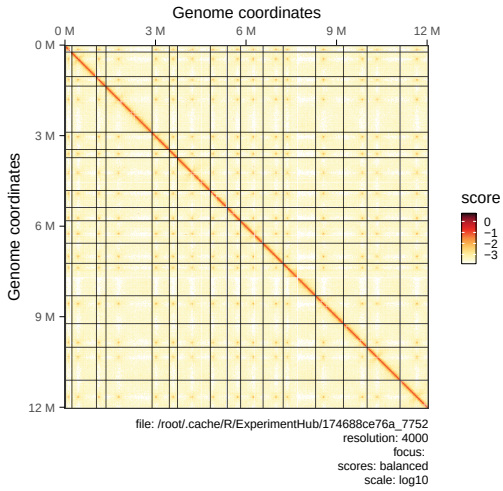
```
plotMatrix(hic, compare.to = hic2)
```



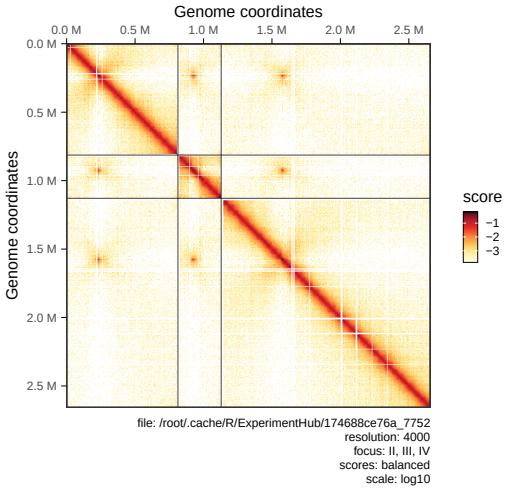
4.1.4 Plotting multiple chromosomes

Interactions from multiple chromosomes can be visualized in a Hi-C heatmap. One needs to (1) first parse the entire contact matrix in `R`, (2) then subset interactions over chromosomes of interest with `[` and (3) use `plotMatrix` to generate the multi-chromosome plot.

```
full_hic <- import(cf, resolution = 4000)
plotMatrix(full_hic)
```




```
hic_subset <- full_hic[c("II", "III", "IV")]
plotMatrix(hic_subset)
```



4.2 Hi-C maps customization options

A number of customization options are available for the `plotMatrix` function. The next subsections focus on how to:

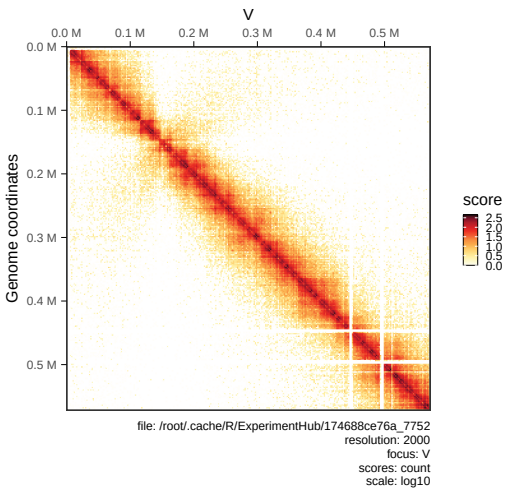
- Pick the `scores` of interest to represent in a Hi-C heatmap;
- Change the numeric scale and boundaries;
- Change the color map;
- Extra customization options

4.2.1 Choosing scores

By default, `plotMatrix` will attempt to plot `balanced` (coverage normalized) Hi-C matrices. However, extra scores may be associated with interactions in a `HiCExperiment` object (more on this in the next chapter)

For instance, we can plot the `count` scores, which are unnormalized raw contact counts directly obtained when binning a `.pairs` file:

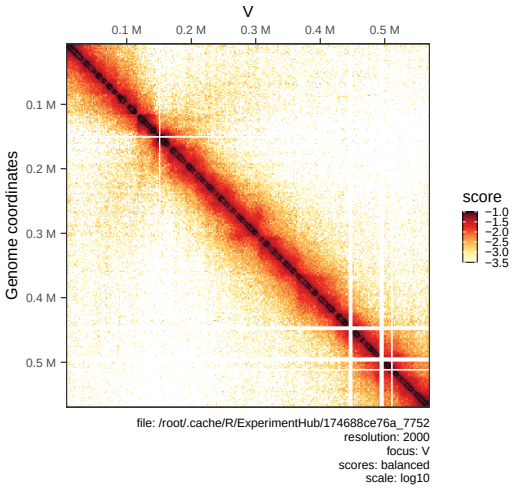
```
plotMatrix(hic, use.scores = 'count')
```



4.2.2 Choosing scale

The color scale is automatically adjusted to range from the minimum to the maximum `scores` of the `HiCExperiment` being plotted. This can be adjusted using the `limits` argument.

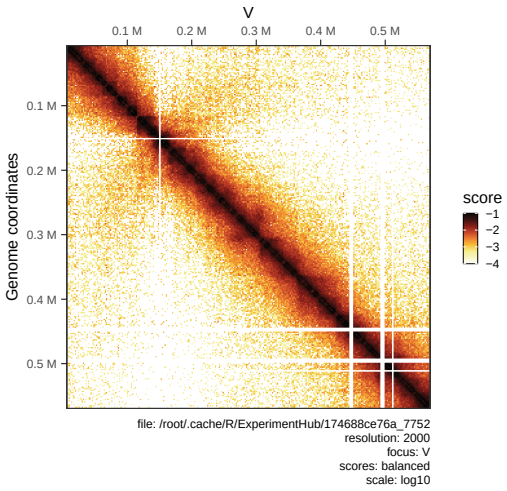
```
plotMatrix(hic, limits = c(-3.5, -1))
```



4.2.3 Choosing color map

`?HiContacts::palettes` returns a list of available color maps to use with `plotMatrix`. Any custom color map can also be used by manually specifying a vector of colors.

```
# ----- `afmhotr` color map is shipped in the `HiContacts` package
afmhotrColors()
##  [1] "#ffffff" "#f8f5c3" "#f4ee8d" "#f6be35" "#ee7d32" "#c44228" "#821d19"
##  [8] "#381211" "#050606"
plotMatrix(
    hic,
    use.scores = 'balanced',
    limits = c(-4, -1),
    cmap = afmhotrColors()
)
```



4.3 Advanced visualization

4.3.1 Overlaying topological features

Topological features (e.g. chromatin loops, domain borders, A/B compartments, e.g. …) are often displayed over a Hi-C heatmap.

To illustrate how to do this, let's import pre-computed chromatin loops in `R`. These loops have been identified using `chromosight` (@Matthey_Doret_2020) on the contact matrix which we imported interactions from.

```
library(rtracklayer)
library(InteractionSet)
loops <- system.file('extdata', 'S288C-loops.bedpe', package = 'HiCExperiment') |>
    import() |>
    makeGInteractionsFromGRangesPairs()
loops
##  GInteractions object with 162 interactions and 0 metadata columns:
##          seqnames1       ranges1     seqnames2       ranges2
##              <Rle>     <IRanges>         <Rle>     <IRanges>
##      [1]         I     3001-4000 ---         I   29001-30000
##      [2]         I   29001-30000 ---         I   50001-51000
```

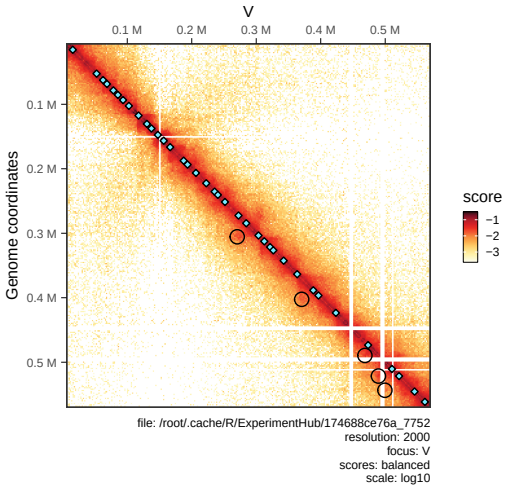
```
##      [3]         I   95001-96000 ---         I 128001-129000
##      [4]         I 133001-134000 ---         I 157001-158000
##      [5]        II     8001-9000 ---        II   46001-47000
##      ...       ...           ... ...       ...           ...
##    [158]       XVI 773001-774000 ---       XVI 803001-804000
##    [159]       XVI 834001-835000 ---       XVI 859001-860000
##    [160]       XVI 860001-861000 ---       XVI 884001-885000
##    [161]       XVI 901001-902000 ---       XVI 940001-941000
##    [162]       XVI 917001-918000 ---       XVI 939001-940000
##    -------
##    regions: 316 ranges and 0 metadata columns
##    seqinfo: 16 sequences from an unspecified genome; no seqlengths
```

Similarly, borders have also been mapped with `chromosight`.
We can also import them in `R`.

```
borders <- system.file('extdata', 'S288C-borders.bed', package = 'HiCExperiment') |>
    import()
borders
##  GRanges object with 814 ranges and 0 metadata columns:
##          seqnames        ranges strand
##             <Rle>     <IRanges>  <Rle>
##      [1]         I   73001-74000      *
##      [2]         I 108001-109000      *
##      [3]         I 181001-182000      *
##      [4]        II   90001-91000      *
##      [5]        II 119001-120000      *
##      ...       ...           ...    ...
##    [810]       XVI 777001-778000      *
##    [811]       XVI 796001-797000      *
##    [812]       XVI 811001-812000      *
##    [813]       XVI 890001-891000      *
##    [814]       XVI 933001-934000      *
##    -------
##    seqinfo: 16 sequences from an unspecified genome; no seqlengths
```

Chromatin loops are stored in `GInteractions` while borders are `GRanges`. The former will be displayed as **off-diagonal circles** and the later as **on-diagonal diamonds** on the Hi-C heatmap.

```
plotMatrix(hic, loops = loops, borders = borders)
```



4.3.2 Aggregated Hi-C maps

Finally, Hi-C map "snippets" (i.e. extracts) are often aggregated together to show an average signal. This analysis is sometimes referred to as APA (Aggregated Plot Analysis).

Aggregated Hi-C maps can be computed over a collection of `targets` using the `aggregate` function. These targets can be `GRanges` (to extract on-diagonal snippets) or `GInteractions` (to extract off-diagonal snippets). The `flankingBins` specifies how many matrix bins should be extracted on each side of the `targets` of interest.

Here, we compute the aggregated Hi-C snippets of ± 15kb around each chromatin loop listed in `loops`.

```
hic <- zoom(hic, 1000)
aggr_loops <- aggregate(hic, targets = loops, flankingBins = 15)
##  Going through preflight checklist...
##  Parsing the entire contact matrice as a sparse matrix...
##  Modeling distance decay...
##  Filtering for contacts within provided targets...
```

```
aggr_loops
##  `AggrHiCExperiment` object over 148 targets
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: 148 targets
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 1000
##  interactions: 961
##  scores(4): count balanced expected detrended
##  slices(4): count balanced expected detrended
##  topologicalFeatures: targets(148) compartments(0) borders(0) loops(0) viewpoints(0) cent
##  pairsFile: N/A
##  metadata(0):
```

`aggregate` generates a `AggrHiCExperiment` object, a flavor of `HiCExperiment` class of objects.

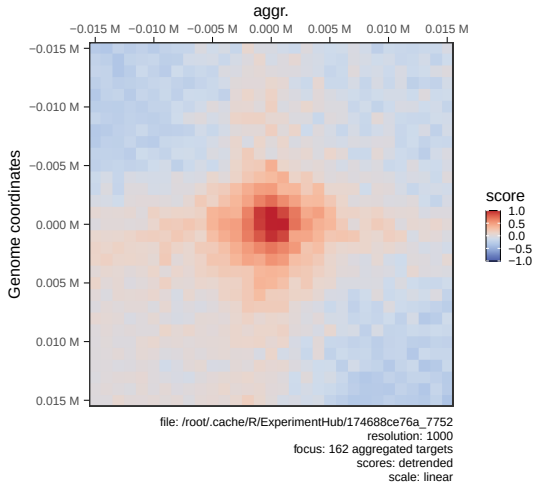
- `AggrHiCExperiment` objects have an extra `slices` slot. This stores a list of `array`s, one per `scores`. Each `array` is of 3 dimensions, `x` and `y` representing the heatmap axes, and `z` representing the index of the `target`.
- `AggrHiCExperiment` objects also have a mandatory `topologicalFeatures` element named `targets`, storing the genomic loci provided in `aggregate`.

```
slices(aggr_loops)
##  List of length 4
##  names(4): count balanced expected detrended
dim(slices(aggr_loops, 'count'))
##  [1]  31  31 148
topologicalFeatures(aggr_loops, 'targets')
##  Pairs object with 148 pairs and 0 metadata columns:
##                      first            second
##                  <GRanges>         <GRanges>
##      [1]     I:14501-44500     I:35501-65500
##      [2]    I:80501-110500   I:113501-143500
##      [3]   I:118501-148500   I:142501-172500
##      [4]    II:33501-63500    II:63501-93500
##      [5]  II:134501-164500  II:159501-189500
##      ...               ...               ...
```

```
##     [144] XVI:586501-616500 XVI:606501-636500
##     [145] XVI:733501-763500 XVI:754501-784500
##     [146] XVI:758501-788500 XVI:788501-818500
##     [147] XVI:819501-849500 XVI:844501-874500
##     [148] XVI:845501-875500 XVI:869501-899500
```

The resulting `AggrHiCExperiment` can be plotted using the same `plotMatrix` function with the arguments described above.

```
plotMatrix(
    aggr_loops,
    use.scores = 'detrended',
    scale = 'linear',
    limits = c(-1, 1),
    cmap = bgrColors()
)
```



References

Part II

In-depth Hi-C analysis

5 Matrix-centric analysis

> **Aims**
>
> This chapter focuses on the various analytical tools offered by `HiContacts` to compute matrix-related metrics from a `HiCExperiment` object.

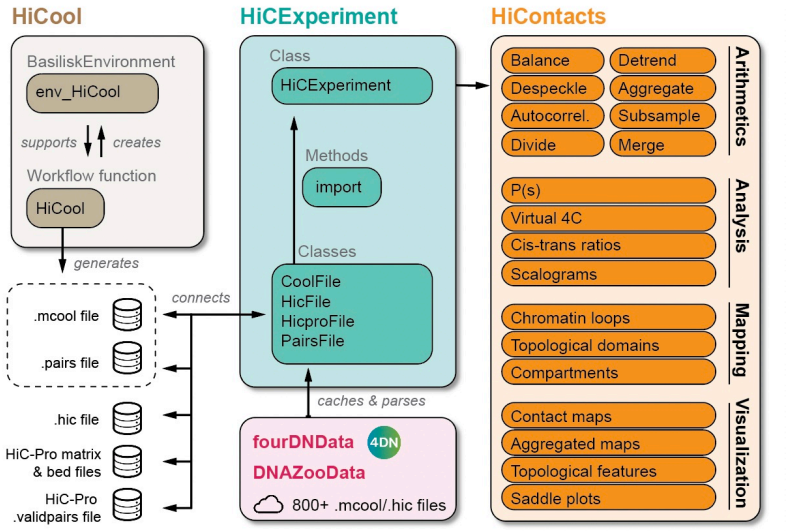
In the first part of this book, we have seen how to query parts or all of the data contained in Hi-C contact matrices using the `HiCExperiment` object (Chapter 2), how to manipulate `HiCExperiment` objects (Chapter 3) and how to visualize Hi-C contact matrices as heatmaps (Chapter 4).

The `HiContacts` package directly operates on `HiCExperiment` objects and extends its usability by providing a comprehensive toolkit to analyze Hi-C data, focusing on four main topics:

- Contact matrix-centric analyses (this chapter)
- Interactions-centric analyses (Chapter 6)
- Structural feature annotations (Chapter 7)
- Hi-C visualization (see previous chapter)

Matrix-centric analyses consider a `HiCExperiment` object from the "matrix" perspective to perform a range of matrix-based operations. This encompasses:

- Computing observed/expected (O/E) map
- Computing auto-correlation map
- Smoothing out a contact map
- Merging multiple Hi-C maps together
- Comparing two Hi-C maps to each other



Note

- All the functions described in this chapter are **endomorphisms**: they take `HiCExperiment` objects as input and return modified `HiCExperiment` objects.
- Internally, most of the functions presented in this chapter make a call to `as.matrix(<HiCExperiment>)` to coerce it into a `matrix`.

Generating the example `hic` object

To demonstrate `HiContacts` functionalities, we will create an `HiCExperiment` object from an example `.cool` file provided in the `HiContactsData` package.

```r
library(HiCExperiment)
library(HiContactsData)

# ---- This downloads an example `.mcool` file and caches it locally
coolf <- HiContactsData('yeast_wt', 'mcool')

# ---- This creates a connection to the disk-stored `.mcool` file
cf <- CoolFile(coolf)
cf

# ---- This imports contacts from the chromosome `II` at resolution `2000`
hic <- import(cf, focus = 'II', resolution = 2000)
```

```r
hic
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 34063
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

5.1 Operations in an individual matrix

5.1.1 Balancing a raw interaction count map

Hi-C sequencing coverage is systematically affected by multiple confounding factors, e.g. density of restriction sites, GC%, genome mappability, etc.. Overall, it generally ends up not homogenous throughout the entire genome and this leads to artifacts in un-normalized `count` matrices.

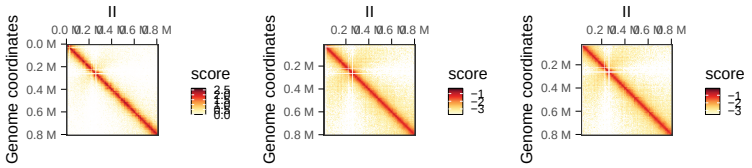
To correct for sequencing coverage heterogeneity of raw `count`

maps, Hi-C data can be normalized using matrix balancing approaches (@Cournac_2012, @Imakaev_2012). This is generally done directly on the disk-stored matrices using out-of-memory strategies (e.g. with `cooler balance <.cool>`). However, if contact matrix files are imported into a `HiCExperiment` object but no `balanced` scores are available, in-memory balancing can be performed using the `normalize` function. This adds an extra `ICE` element in `scores` list (while the `interactions` themselves are unmodified).

```
normalized_hic <- normalize(hic)
normalized_hic
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 34063
##  scores(3): count balanced ICE
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

It is possible to plot the different `scores` of the resulting object to visualize the newly computed `scores`. In this example, `ICE` scores should be nearly identical to `balanced` scores, which were originally imported from the disk-stored contact matrix.

```
patchwork::wrap_plots(
    plotMatrix(normalized_hic, use.scores = 'count', caption = FALSE),
    plotMatrix(normalized_hic, use.scores = 'balanced', caption = FALSE),
    plotMatrix(normalized_hic, use.scores = 'ICE', caption = FALSE),
    nrow = 1
)
```



5.1.2 Computing observed/expected (O/E) map

The most prominent feature of a balanced Hi-C matrix is the strong main diagonal. This main diagonal is observed because interactions between immediate adjacent genomic loci are more prone to happen than interactions spanning longer genomic distances. This "expected" behavior is due to the polymer nature of the chromosomes being studied, and can be locally estimated using the distance-dependent interaction frequency (a.k.a. the "distance law", or P(s)). It can be used to compute an `expected` matrix on interactions.

When it is desirable to "mask" this polymer behavior to emphasize topological structures formed by chromosomes, one can divide a given balanced matrix by its `expected` matrix, i.e. calculate the observed/expected (O/E) map. This is sometimes called "detrending", as it effectively removes the average polymer behavior from the balanced matrix.

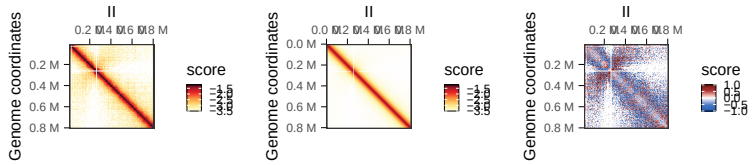
The `detrend` function performs this operation on a given `HiCExperiment` object. It adds two extra elements in `scores` list: `expected` and `detrended` metrics (while the `interactions` themselves are unmodified).

```
detrended_hic <- detrend(hic)
detrended_hic
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
```

```
##  interactions: 34063
##  scores(4): count balanced expected detrended
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

Topological features will be visually more prominent in the O/E `detrended` Hi-C map.

```
patchwork::wrap_plots(
    plotMatrix(detrended_hic, use.scores = 'balanced', scale = 'log10', limits = c(-3.5, -1.
    plotMatrix(detrended_hic, use.scores = 'expected', scale = 'log10', limits = c(-3.5, -1.
    plotMatrix(detrended_hic, use.scores = 'detrended', scale = 'linear', limits = c(-1, 1),
    nrow = 1
)
```



> **Scale for `detrended` scores**
>
> - `expected` scores are in `linear` scale and ± in the same amplitude than `balanced` scores;
> - `detrended` scores are in `log2` scale, in general approximately centered around 0. When plotting `detrended` scores, `scale = linear` should be set to prevent the default `log10` scaling.

5.1.3 Computing autocorrelated map

Correlation matrices are often calculated from balanced Hi-C matrices. For instance, in genomes composed of eu- and het-

erochromatin, a correlation matrix can be used to reveal a checkerboard pattern emphasizing the segregation of chromatin into two A/B compartments (@Lieberman_Aiden_2009).

The `autocorrelate` function is used to compute a correlation matrix of a `HiCExperiment` object. For each pair of interacting loci, the `autocorrelated` score represents the correlation between their respective interaction profiles with the rest of the genome.

```
autocorr_hic <- autocorrelate(hic)
##
autocorr_hic
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 34063
##  scores(5): count balanced expected detrended autocorrelated
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

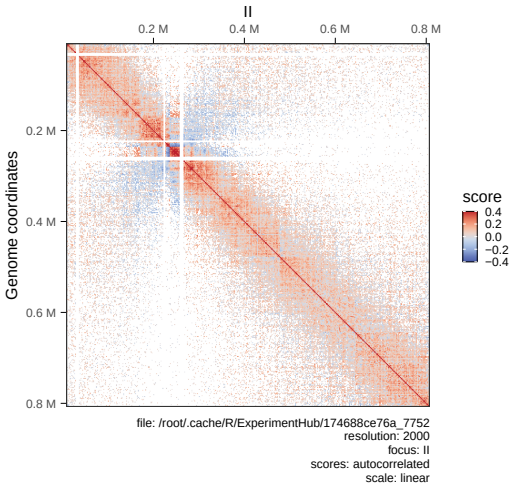
Since these metrics represent correlation scores, they range between `-1` and `1`. Two loci with an `autocorrelated` score close to `-1` have anti-correlated interaction profiles, while two loci with a `autocorrelated` score close to `1` are likely to interact with shared targets.

```
summary(scores(autocorr_hic, 'autocorrelated'))
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -0.4156  0.0025  0.0504  0.0645  0.1036  1.0000     564
```

Correlated and anti-correlated loci will be visually represented in the `autocorrelated` Hi-C map in red and blue pixels, respectively.

> **Note**
>
> Here we have illustrated how to compute an autocorrelation matrix from a `HiCExperiment` object using the example **yeast** Hi-C experiment. Bear in mind that this is unusual and not very useful, as yeast chromatin is not segregated in two compartments but rather follows a Rabl conformation (@Duan_2010). An example of autocorrelation map from a vertebrate Hi-C experiment (for which chromatin is segregated in A/B compartments) is shown in Chapter 10.

```
plotMatrix(
    autocorr_hic,
    use.scores = 'autocorrelated',
    scale = 'linear',
    limits = c(-0.4, 0.4),
    cmap = bgrColors()
)
```



Scale for `autocorrelated` scores

- `autocorrelated` scores are in `linear` scale, in general approximately centered around 0. When plotting `autocorrelated` scores, `scale = linear` should be set to prevent the default `log10` scaling.
- `limits` should be manually set to `c(-x, x)` (`0 < x <= 1`) to ensure that the color range is effectively centered on `0`.

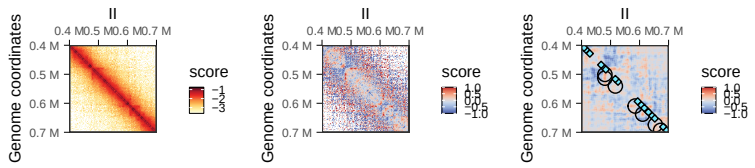
5.1.4 Despeckling (smoothing out) a contact map

Shallow-sequenced Hi-C libraries or matrices binned with an overly small bin size sometimes produce "grainy" Hi-C maps with noisy backgrounds. A grainy map may also be obtained when dividing two matrices, e.g. when computing the O/E ratio with `detrend`. This is particularly true for sparser long-range interactions. To overcome such limitations, `HiCExperiment` objects can be "`despeckled`" to smooth out focal speckles.

```
hic2 <- detrend(hic['II:400000-700000'])
hic2 <- despeckle(hic2, use.scores = 'detrended', focal.size = 2)
hic2
##  `HiCExperiment` object with 168,785 contacts over 150 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II:400,000-700,000"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 11325
##  scores(5): count balanced expected detrended detrended.despeckled
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

The added `<use.scores>.despeckled` scores correspond to scores averaged using a window, whose width is provided with the `focal.size` argument. This results in a smoother Hi-C heatmap, effectively removing the "speckles" observed at longer range.

```
library(InteractionSet)
loops <- system.file('extdata', 'S288C-loops.bedpe', package = 'HiCExperiment') |>
    import() |>
    makeGInteractionsFromGRangesPairs()
borders <- system.file('extdata', 'S288C-borders.bed', package = 'HiCExperiment') |>
    import()
patchwork::wrap_plots(
    plotMatrix(hic2, caption = FALSE),
    plotMatrix(hic2, use.scores = 'detrended', scale = 'linear', limits = c(-1, 1), caption
    plotMatrix(
        hic2,
        use.scores = 'detrended.despeckled',
        scale = 'linear',
        limits = c(-1, 1),
        caption = FALSE,
        loops = loops,
        borders = borders
    ),
    nrow = 1

)
```



> **Scale for `despeckled` scores**
>
> `despeckled` scores are in the same scale than the `scores` they were computed from.

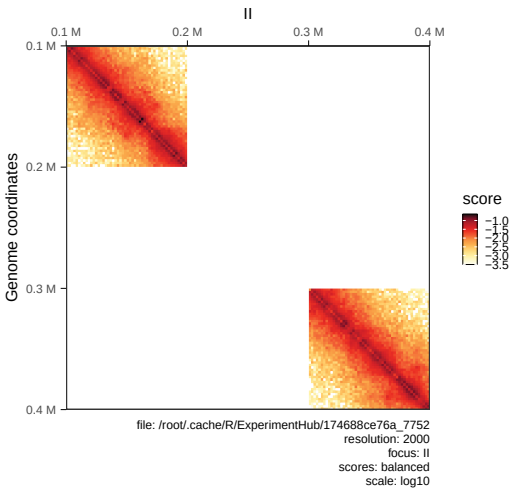
5.2 Operations between multiple matrices

5.2.1 Merging maps

Hi-C libraries are often sequenced in multiple rounds, for example when high genome coverage is required. This results in multiple contact matrix files being generated. The `merge` function can be used to bind several `HiCExperiment` objects into a single one.

The different `HiCExperiment` objects do not need to all have identical `regions`, as shown in the following example.

```
hic_sub1 <- subsetByOverlaps(hic, GRanges("II:100001-200000"))
hic_sub2 <- subsetByOverlaps(hic, GRanges("II:300001-400000"))
bound_hic <- merge(hic_sub1, hic_sub2)
plotMatrix(bound_hic)
```



5.2.2 Computing ratio between two maps

Comparing two Hi-C maps can be useful to infer which genomic loci are differentially interacting between experimental conditions. Comparing two `HiCExperiment` objects can be done in `R` using the `divide` function.

For example, we can divide the *eco1* mutant Hi-C data by wild-type Hi-C dataset using the `divide` function.

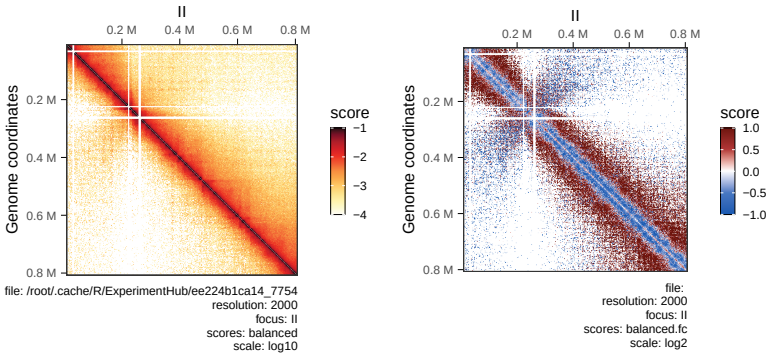
```
hic_eco1 <- import(
    CoolFile(HiContactsData('yeast_eco1', 'mcool')),
    focus = 'II',
    resolution = 2000
)
```

```
div_contacts <- divide(hic_eco1, by = hic)
div_contacts
##  `HiCExperiment` object with 996,154 contacts over 407 regions
##  -------
##  fileName: N/A
##  focus: "II"
##  resolutions(1): 2000
##  active resolution: 2000
##  interactions: 60894
##  scores(6): count.x balanced.x count.by balanced.by balanced.fc balanced.l2fc
##  topologicalFeatures: ()
##  pairsFile: N/A
##  metadata(2): hce_list operation
```

We can visually compare wild-type and *eco1* maps side by side (left) and their ratio map (right). This highlights the depletion of short-range and increase of long-range interactions in the *eco1* dataset.

```
cowplot::plot_grid(
    plotMatrix(hic_eco1, compare.to = hic, limits = c(-4, -1)),
    plotMatrix(
        div_contacts,
        use.scores = 'balanced.fc',
```

```
        scale = 'log2',
        limits = c(-1, 1),
        cmap = bwrColors()
    )
)
```



References

6 Interactions-centric analysis

> **Aims**
>
> This chapter focuses on the various analytical tools offered by `HiContacts` to compute interaction-related metrics from a `HiCExperiment` object.

Interaction-centric analyses consider a `HiCExperiment` object from the "interactions" perspective to perform a range of operations on genomic interactions.
This encompasses:

- Computing the "distance law" (a.k.a. P(s)), i.e. the distance-dependent interaction frequency
- Computing profiles of interactions between a locus of interest and the rest of the genome, (a.k.a. virtual 4C profiles)
- Computing cis/trans interaction ratios
- Computing distribution of distance-dependent interaction frequency along chromosomes, a.k.a. scalograms

> **Note**
>
> - Contrary to functions presented in the previous chapter, the functions described in this chapter are **not**: they take `HiCExperiment` objects as input and generally return data frames rather than modified `HiCExperiment` objects.
> - Internally, most of the functions presented in this chapter make a call to `interactions(<HiCExperiment>)` to coerce it into `GInteractions`.

Generating the example `hic` object

To demonstrate `HiContacts` functionalities, we will create an `HiCExperiment` object from an example `.cool` file provided in the `HiContactsData` package.

```
library(HiCExperiment)
library(HiContactsData)

# ---- This downloads example `.mcool` and `.pairs` files and caches them locally
coolf <- HiContactsData('yeast_wt', 'mcool')
pairsf <- HiContactsData('yeast_wt', 'pairs.gz')

# ---- This creates a connection to the disk-stored `.mcool` file
cf <- CoolFile(coolf)
cf

# ---- This creates a connection to the disk-stored `.pairs` file
pf <- PairsFile(pairsf)
pf

# ---- This imports contacts from the chromosome `II` at resolution `2000`
hic <- import(cf, focus = 'II', resolution = 2000)
```

```
hic
##  `HiCExperiment` object with 471,364 contacts over 407 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 2000
##  interactions: 34063
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) centromeres(16)
##  pairsFile: N/A
##  metadata(0):
```

6.1 Distance law(s)

6.1.1 P(s) from a single `.pairs` file

Distance laws are generally computed directly from `.pairs` files. This is because the `.pairs` files are at 1-bp resolution whereas the contact matrices (for example from `.cool` files) are binned at a minimum resolution.

An example `.pairs` file can be fetched from the `ExperimentHub` database using the `HiContactsData` package.

```
library(HiCExperiment)
library(HiContactsData)
pairsf <- HiContactsData('yeast_wt', 'pairs.gz')
pf <- PairsFile(pairsf)
```

```
pf
##  PairsFile object
##  resource: /root/.cache/R/ExperimentHub/174ff8a7b2_7753
```

If needed, `PairsFile` connections can be imported directly into a `GInteractions` object with `import()`.

```
import(pf)
##  GInteractions object with 471364 interactions and 3 metadata columns:
##             seqnames1   ranges1     seqnames2   ranges2 |     frag1     frag2
##                 <Rle> <IRanges>         <Rle> <IRanges> | <numeric> <numeric>
##         [1]        II       105 ---        II     48548 |      1358      1681
##         [2]        II       113 ---        II     45003 |      1358      1658
##         [3]        II       119 ---        II    687251 |      1358      5550
##         [4]        II       160 ---        II     26124 |      1358      1510
##         [5]        II       169 ---        II     39052 |      1358      1613
##         ...       ...       ... ...       ...       ... .       ...       ...
##    [471360]        II    808605 ---        II    809683 |      6316      6320
##    [471361]        II    808609 ---        II    809917 |      6316      6324
##    [471362]        II    808617 ---        II    809506 |      6316      6319
##    [471363]        II    809447 ---        II    809685 |      6319      6321
##    [471364]        II    809472 ---        II    809675 |      6319      6320
##             distance
```

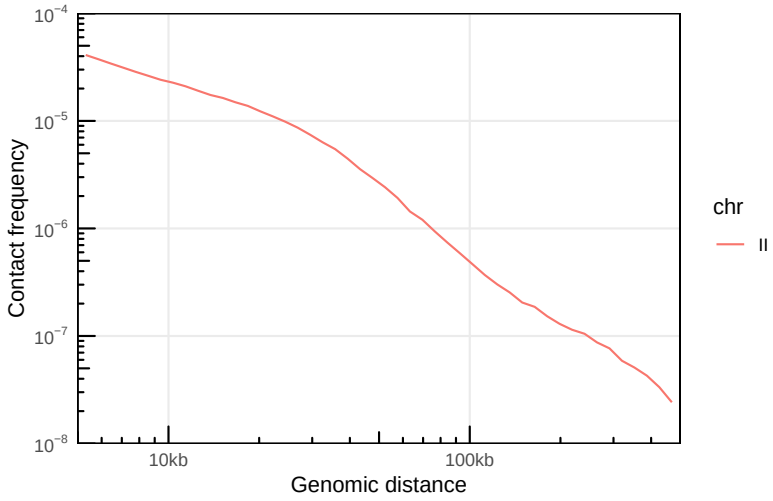
```
##             <integer>
##         [1]     48443
##         [2]     44890
##         [3]    687132
##         [4]     25964
##         [5]     38883
##         ...       ...
##    [471360]      1078
##    [471361]      1308
##    [471362]       889
##    [471363]       238
##    [471364]       203
##    -------
##    regions: 549331 ranges and 0 metadata columns
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We can compute a P(s) per chromosome from this `.pairs` file using the `distanceLaw` function.

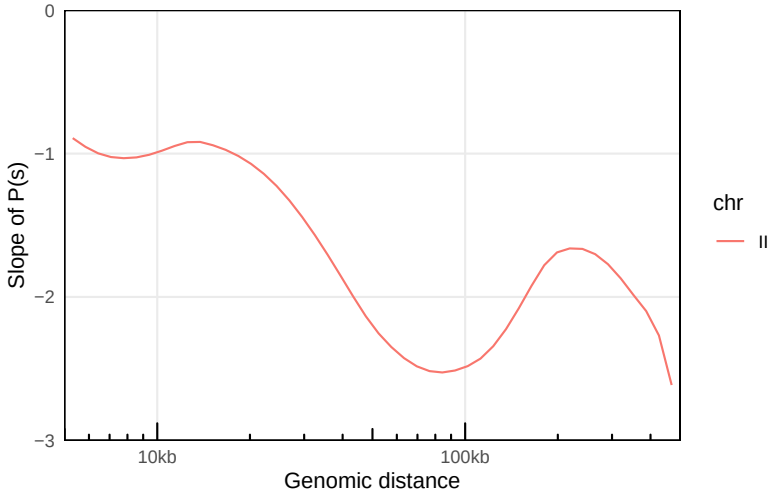
```
library(HiContacts)
ps <- distanceLaw(pf, by_chr = TRUE)
##  Importing pairs file /root/.cache/R/ExperimentHub/174ff8a7b2_7753 in memory. This may ta
ps
##  # A tibble: 115 x 6
##    chr   binned_distance          p     norm_p norm_p_unity slope
##    <chr>           <dbl>      <dbl>      <dbl>        <dbl> <dbl>
##  1 II                 14 0.00000212 0.00000106         2.27  0
##  2 II                 16 0.0000170  0.0000170         36.4   1.56
##  3 II                 17 0.0000361  0.0000180         38.6   1.55
##  4 II                 19 0.0000424  0.0000212         45.5   1.55
##  5 II                 21 0.0000467  0.0000233         50.0   1.54
##  6 II                 23 0.0000870  0.0000290         62.1   1.53
##  # i 109 more rows
```

The `plotPs()` and `plotPsSlope()` functions are convenient `ggplot2`-based functions with pre-configured settings optimized for P(s) visualization.

```
library(ggplot2)
plotPs(ps, aes(x = binned_distance, y = norm_p, color = chr))
##  Warning: Removed 67 rows containing missing values (`geom_line()`).
```




```
plotPsSlope(ps, aes(x = binned_distance, y = slope, color = chr))
##  Warning: Removed 67 rows containing missing values (`geom_line()`).
```



6.1.2 P(s) for multiple `.pairs` files

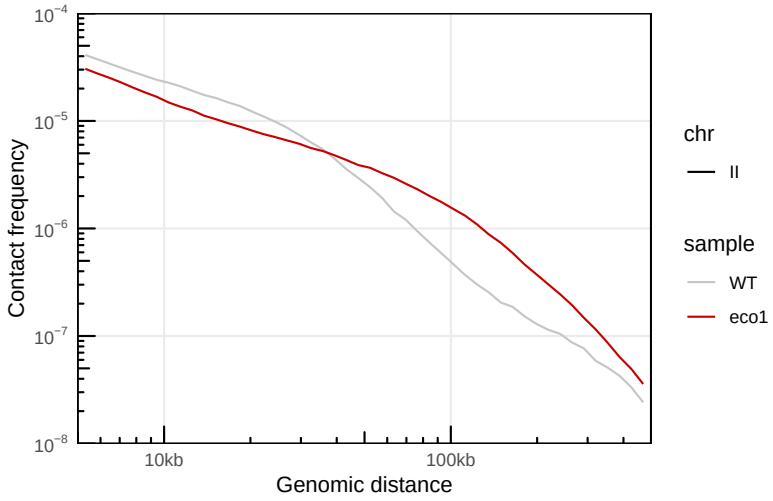
Let's first import a second example dataset. We'll import pairs identified in a *eco1* yeast mutant.

```
eco1_pairsf <- HiContactsData('yeast_eco1', 'pairs.gz')
eco1_pf <- PairsFile(eco1_pairsf)
```

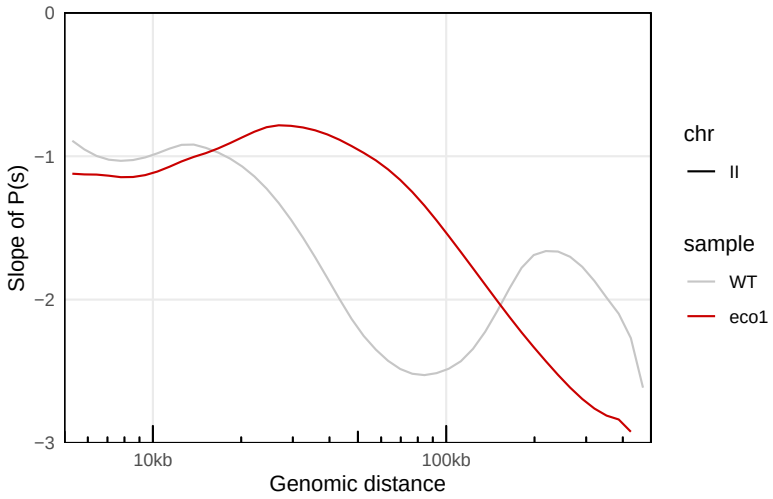
```
eco1_ps <- distanceLaw(eco1_pf, by_chr = TRUE)
##  Importing pairs file /root/.cache/R/ExperimentHub/fae489c8d53_7755 in memory. This may t
eco1_ps
##  # A tibble: 115 x 6
##    chr   binned_distance          p     norm_p norm_p_unity slope
##    <chr>           <dbl>      <dbl>      <dbl>        <dbl> <dbl>
##  1 II                 14 0.00000201 0.00000100        0.660  0
##  2 II                 16 0.0000221  0.0000221        14.5    1.46
##  3 II                 17 0.0000492  0.0000246        16.2    1.46
##  4 II                 19 0.0000412  0.0000206        13.5    1.45
##  5 II                 21 0.0000653  0.0000326        21.5    1.45
##  6 II                 23 0.0000803  0.0000268        17.6    1.44
##  # i 109 more rows
```

A little data wrangling can help plotting the distance laws for 2 different samples in the same plot.

```
library(dplyr)
merged_ps <- rbind(
    ps |> mutate(sample = 'WT'),
    eco1_ps |> mutate(sample = 'eco1')
)
plotPs(merged_ps, aes(x = binned_distance, y = norm_p, color = sample, linetype = chr)) +
    scale_color_manual(values = c('#c6c6c6', '#ca0000'))
##  Warning: Removed 134 rows containing missing values (`geom_line()`).
```




```
plotPsSlope(merged_ps, aes(x = binned_distance, y = slope, color = sample, linetype = chr))
    scale_color_manual(values = c('#c6c6c6', '#ca0000'))
##  Warning: Removed 135 rows containing missing values (`geom_line()`).
```

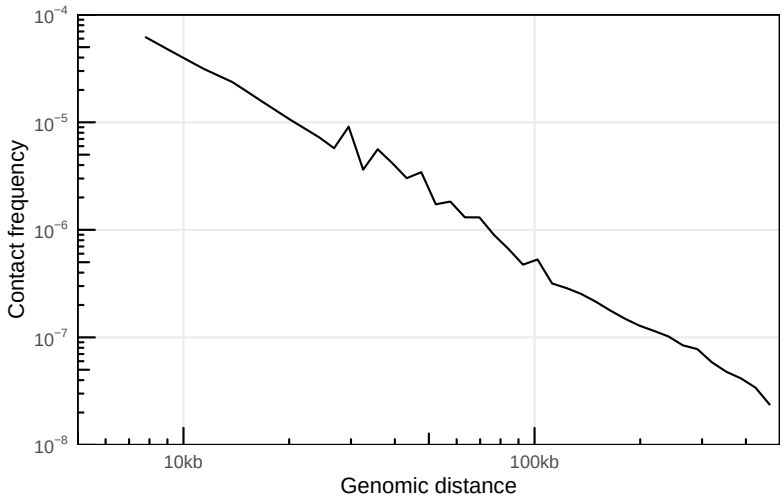


6.1.3 P(s) from `HiCExperiment` objects

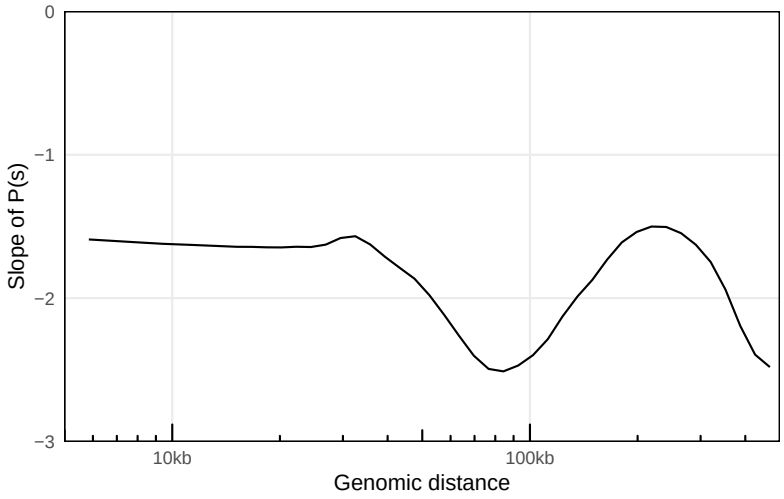
Alternatively, distance laws can be computed from binned matrices directly by providing `HiCExperiment` objects. For deeply sequenced datasets, this can be significantly faster than when

using original `.pairs` files, but the smoothness of the resulting curves will be greatly impacted, notably at short distances.

```
ps_from_hic <- distanceLaw(hic, by_chr = TRUE)
##  pairsFile not specified. The P(s) curve will be an approximation.
plotPs(ps_from_hic, aes(x = binned_distance, y = norm_p))
##  Warning: Removed 9 rows containing missing values (`geom_line()`).
```




```
plotPsSlope(ps_from_hic, aes(x = binned_distance, y = slope))
##  Warning: Removed 8 rows containing missing values (`geom_line()`).
```



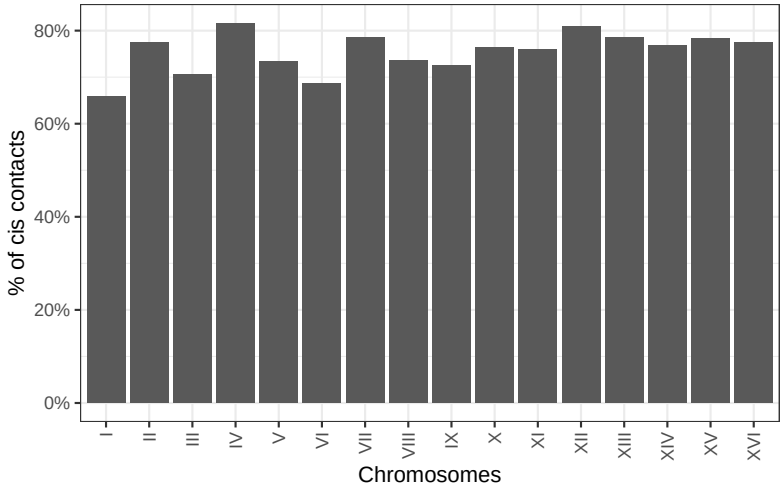
6.2 Cis/trans ratios

The ratio between cis interactions and trans interactions is often used to assess the overall quality of a Hi-C dataset. It can be computed *per chromosome* using the `cisTransRatio()` function. You will need to provide a **genome-wide** `HiCExperiment` to estimate cis/trans ratios!

```
full_hic <- import(cf, resolution = 2000)
ct <- cisTransRatio(full_hic)
ct
##  # A tibble: 16 x 6
##  # Groups:   chr [16]
##    chr       cis  trans n_total cis_pct trans_pct
##    <fct>   <dbl>  <dbl>   <dbl>   <dbl>     <dbl>
##  1 I      186326  96738  283064   0.658     0.342
##  2 II     942728 273966 1216694   0.775     0.225
##  3 III    303980 127087  431067   0.705     0.295
##  4 IV    1858062 418218 2276280   0.816     0.184
##  5 V      607090 220873  827963   0.733     0.267
##  6 VI     280282 127771  408053   0.687     0.313
##  # i 10 more rows
```

It can be plotted using `ggplot2`-based visualization functions.

```
ggplot(ct, aes(x = chr, y = cis_pct)) +
    geom_col(position = position_stack()) +
    theme_bw() +
    guides(x=guide_axis(angle = 90)) +
    scale_y_continuous(labels = scales::percent) +
    labs(x = 'Chromosomes', y = '% of cis contacts')
```



Cis/trans contact ratios will greatly vary **depending on the cell cycle phase the sample is in!** For instance, chromosomes during the mitosis phase of the cell cycle have very little trans contacts, due to their structural organization and individualization.

6.3 Virtual 4C profiles

Interaction profile of a genomic locus of interest with its surrounding environment or the rest of the genome is frequently generated. In some cases, this can help in identifying and/or comparing regulatory or structural interactions.

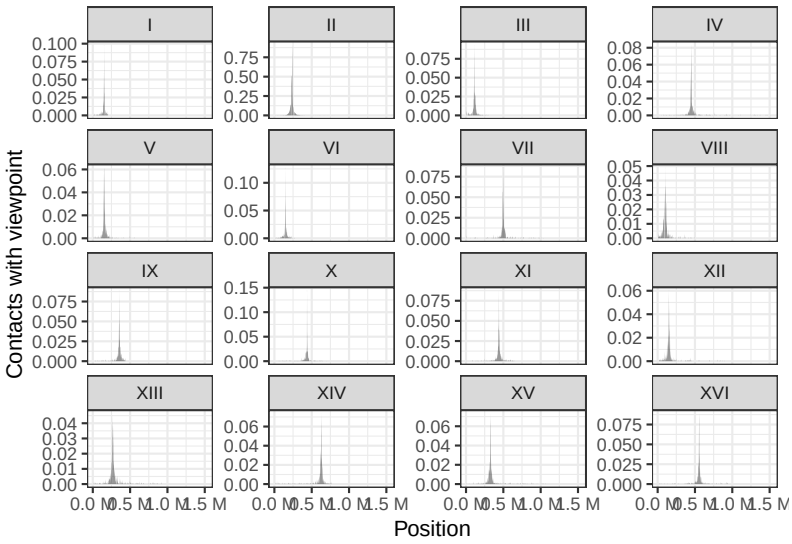
For instance, we can compute the genome-wide virtual 4C profile of interactions anchored at the centromere in chromosome `II` (located at ~ 238kb).

```
library(GenomicRanges)
v4C <- virtual4C(full_hic, viewpoint = GRanges("II:230001-240000"))
v4C
##  GRanges object with 6045 ranges and 4 metadata columns:
##            seqnames        ranges strand |       score        viewpoint
##               <Rle>     <IRanges>  <Rle> |   <numeric>      <character>
##       [1]        I        1-2000      * |  0.00000000 II:230001-240000
##       [2]        I     2001-4000      * |  0.00000000 II:230001-240000
##       [3]        I     4001-6000      * |  0.00129049 II:230001-240000
##       [4]        I     6001-8000      * |  0.00000000 II:230001-240000
##       [5]        I    8001-10000      * |  0.00000000 II:230001-240000
##       ...      ...           ...    ... .         ...              ...
##    [6041]      XVI 940001-942000      * | 0.000775721 II:230001-240000
##    [6042]      XVI 942001-944000      * | 0.000000000 II:230001-240000
##    [6043]      XVI 944001-946000      * | 0.000000000 II:230001-240000
##    [6044]      XVI 946001-948000      * | 0.000000000 II:230001-240000
##    [6045]      XVI 948001-948066      * | 0.000000000 II:230001-240000
##              center in_viewpoint
##           <numeric>    <logical>
##       [1]    1000.5        FALSE
##       [2]    3000.5        FALSE
##       [3]    5000.5        FALSE
##       [4]    7000.5        FALSE
##       [5]    9000.5        FALSE
##       ...       ...          ...
##    [6041]    941000        FALSE
##    [6042]    943000        FALSE
##    [6043]    945000        FALSE
##    [6044]    947000        FALSE
##    [6045]    948034        FALSE
##    -------
##    seqinfo: 16 sequences from an unspecified genome; no seqlengths
```

`ggplot2` can be used to visualize the 4C-like profile over multiple chromosomes.

This clearly highlights trans interactions of the chromosome `II`

```
df <- as_tibble(v4C)

ggplot(df, aes(x = center, y = score)) +

    geom_area(position = "identity", alpha = 0.5) +

    theme_bw() +

    labs(x = "Position", y = "Contacts with viewpoint") +

    scale_x_continuous(labels = scales::unit_format(unit = "M", scale = 1e-06)) +

    facet_wrap(~seqnames, scales = 'free_y')
```

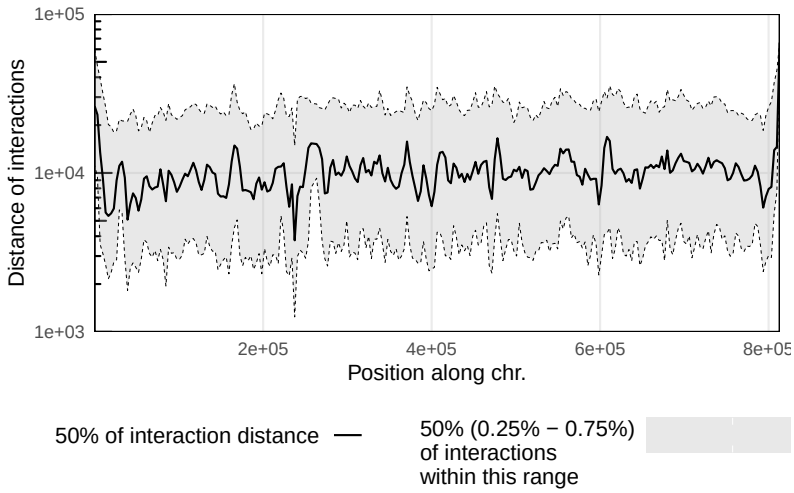


centromere with the centromeres from other chromosomes.

6.4 Scalograms

Scalograms were introduced in @Lioy_2018 to investigate distance-dependent contact frequencies for individual genomic bins along chromosomes.
To generate a scalogram, one needs to provide a `HiCExperiment` object with a valid associated `pairsFile`.

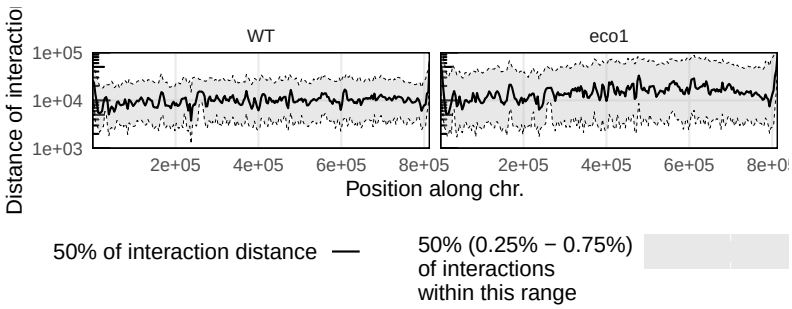
```
pairsFile(hic) <- pairsf
scalo <- scalogram(hic)
##  Importing pairs file /root/.cache/R/ExperimentHub/174ff8a7b2_7753 in memory. This may ta
plotScalogram(scalo |> filter(chr == 'II'), ylim = c(1e3, 1e5))
```



Several scalograms can be plotted together to compare distance-dependent contact frequencies along a given chromosome in different samples.

This example points out the overall longer interactions within the long arm of the chromosome `II` in an `eco1` mutant.

```
eco1_hic <- import(
    CoolFile(HiContactsData('yeast_eco1', 'mcool')),
    focus = 'II',
    resolution = 2000
)

##  see ?HiContactsData and browseVignettes('HiContactsData') for documentation
##  loading from cache
eco1_pairsf <- HiContactsData('yeast_eco1', 'pairs.gz')
##  see ?HiContactsData and browseVignettes('HiContactsData') for documentation
##  loading from cache

pairsFile(eco1_hic) <- eco1_pairsf
eco1_scalo <- scalogram(eco1_hic)
##  Importing pairs file /root/.cache/R/ExperimentHub/fae489c8d53_7755 in memory. This may t
merged_scalo <- rbind(
    scalo |> mutate(sample = 'WT'),

    eco1_scalo |> mutate(sample = 'eco1')
)
plotScalogram(merged_scalo |> filter(chr == 'II'), ylim = c(1e3, 1e5)) +
    facet_grid(~sample)
```



References

7 Finding topological features in Hi-C

> **Aims**
>
> This chapter focuses on the annotation of topological features from Hi-C contact maps, including:
>
> - Chromosome compartments
> - Topologically associating domains
> - Stable chromatin loops

7.1 Chromosome compartments

Chromosome compartments refer to the segregation of the chromatin into active euchromatin (A compartments) and regulated heterochromatin (B compartment).

7.1.1 Importing Hi-C data

To investigate chromosome compartments, we will fetch a contact matrix generated from a micro-C experiment (from @Krietenstein_2020). A subset of the genome-wide dataset is provided in the `OHCA` package. It contains intra-chromosomal interactions within `chr17`, binned at `5000`, `100000` and `250000` bp.

```
library(HiCExperiment)
library(OHCA)
cf <- fs::path_package('OHCA', 'extdata', 'chr17.mcool')
microC <- import(cf, resolution = 250000)
```

```
microC
##  `HiCExperiment` object with 10,086,710 contacts over 334 regions
##  -------
##  fileName: "/tmp/RtmpEEUcH8/Rinst555f2f89bc/OHCA/extdata/chr17.mcool"
##  focus: "whole genome"
##  resolutions(3): 5000 100000 250000
##  active resolution: 250000
##  interactions: 52755
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(0):

seqinfo(microC)
##  Seqinfo object with 1 sequence from an unspecified genome:
##    seqnames seqlengths isCircular genome
##    chr17      83257441         NA   <NA>
```

7.1.2 Annotating A/B compartments

The consensus approach to annotate A/B compartments is to compute the eigenvectors of a Hi-C contact matrix and identify the eigenvector representing the chromosome-wide bi-partite segmentation of the genome.

The `getCompartments()` function performs several internal operations to achieve this:

1. Obtains cis interactions per chromosome
2. Computes O/E contact matrix scores
3. Computes 3 first eigenvectors of this Hi-C contact matrix
4. Normalizes eigenvectors
5. Picks the eigenvector that has the greatest absolute correlation with a phasing track (e.g. a GC% track automatically computed from a genome reference sequence, or a gene density track)
6. Signs this eigenvector so that positive values represent the A compartment

```
phasing_track <- BSgenome.Hsapiens.UCSC.hg38::BSgenome.Hsapiens.UCSC.hg38
microC_compts <- getCompartments(microC, genome = phasing_track)
##  Going through preflight checklist...
##  Parsing intra-chromosomal contacts for each chromosome...
##  Computing eigenvectors for each chromosome...

microC_compts
##  `HiCExperiment` object with 10,086,710 contacts over 334 regions
##  -------
##  fileName: "/tmp/RtmpEEUcH8/Rinst555f2f89bc/OHCA/extdata/chr17.mcool"
##  focus: "whole genome"
##  resolutions(3): 5000 100000 250000
##  active resolution: 250000
##  interactions: 52755
##  scores(2): count balanced
##  topologicalFeatures: compartments(41) borders(0) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(1): eigens
```

`getCompartments()` is an endomorphism: it returns the original object, enriched with two new pieces of information:

- A `compartments` `topologicalFeatures`:

```
topologicalFeatures(microC_compts, "compartments")
##  GRanges object with 41 ranges and 1 metadata column:
##         seqnames            ranges strand | compartment
##            <Rle>         <IRanges>  <Rle> | <character>
##     [1]    chr17    250001-3000000      * |           A
##     [2]    chr17   3000001-3500000      * |           B
##     [3]    chr17   3500001-5500000      * |           A
##     [4]    chr17   5500001-6500000      * |           B
##     [5]    chr17   6500001-8500000      * |           A
##     ...      ...               ...    ... .         ...
##    [37]    chr17 72750001-73250000      * |           A
##    [38]    chr17 73250001-74750000      * |           B
##    [39]    chr17 74750001-79250000      * |           A
##    [40]    chr17 79250001-79750000      * |           B
##    [41]    chr17 79750001-83250000      * |           A
##    -------
```

```
##    seqinfo: 1 sequence from an unspecified genome
```

- The calculated eigenvectors stored in `metadata`:

```
metadata(microC_compts)$eigens
##  GRanges object with 334 ranges and 9 metadata columns:
##                                    seqnames            ranges strand |
##                                       <Rle>         <IRanges>  <Rle> |
##             chr17.chr17_1_250000    chr17          1-250000      * |
##         chr17.chr17_250001_500000    chr17     250001-500000      * |
##         chr17.chr17_500001_750000    chr17     500001-750000      * |
##        chr17.chr17_750001_1000000    chr17    750001-1000000      * |
##       chr17.chr17_1000001_1250000    chr17   1000001-1250000      * |
##                                ...      ...               ...    ... .
##     chr17.chr17_82250001_82500000    chr17 82250001-82500000      * |
##     chr17.chr17_82500001_82750000    chr17 82500001-82750000      * |
##     chr17.chr17_82750001_83000000    chr17 82750001-83000000      * |
##     chr17.chr17_83000001_83250000    chr17 83000001-83250000      * |
##     chr17.chr17_83250001_83257441    chr17 83250001-83257441      * |
##                                    bin_id     weight   chr    center
##                                 <numeric>  <numeric> <Rle> <integer>
##             chr17.chr17_1_250000         0        NaN chr17    125000
##         chr17.chr17_250001_500000         1 0.00626903 chr17    375000
##         chr17.chr17_500001_750000         2 0.00567190 chr17    625000
##        chr17.chr17_750001_1000000         3 0.00528588 chr17    875000
##       chr17.chr17_1000001_1250000         4 0.00464628 chr17   1125000
##                                ...       ...        ...   ...       ...
##     chr17.chr17_82250001_82500000       329 0.00463044 chr17  82375000
##     chr17.chr17_82500001_82750000       330 0.00486910 chr17  82625000
##     chr17.chr17_82750001_83000000       331 0.00561269 chr17  82875000
##     chr17.chr17_83000001_83250000       332 0.00546433 chr17  83125000
##     chr17.chr17_83250001_83257441       333        NaN chr17  83253721
##                                        E1        E2        E3   phasing
##                                 <numeric> <numeric> <numeric> <numeric>
##             chr17.chr17_1_250000  0.000000  0.000000  0.000000  0.383084
##         chr17.chr17_250001_500000  0.450991  0.653287  0.615300  0.433972
##         chr17.chr17_500001_750000  0.716784  0.707461  0.845033  0.465556
##        chr17.chr17_750001_1000000  0.904423  0.414952  0.864288  0.503592
##       chr17.chr17_1000001_1250000  0.913023  0.266287  0.759016  0.547712
##                                ...       ...       ...       ...       ...
```

```
##    chr17.chr17_82250001_82500000  1.147060  0.239112  1.133498  0.550872
##    chr17.chr17_82500001_82750000  1.106937  0.419647  1.169464  0.513212
##    chr17.chr17_82750001_83000000  0.818990  0.591955  0.850340  0.522432
##    chr17.chr17_83000001_83250000  0.874038  0.503175  0.847926  0.528448
##    chr17.chr17_83250001_83257441  0.000000  0.000000  0.000000  0.000000
##                                       eigen
##                                   <numeric>
##             chr17.chr17_1_250000  0.000000
##        chr17.chr17_250001_500000  0.450991
##        chr17.chr17_500001_750000  0.716784
##       chr17.chr17_750001_1000000  0.904423
##      chr17.chr17_1000001_1250000  0.913023
##                                ...       ...
##    chr17.chr17_82250001_82500000  1.147060
##    chr17.chr17_82500001_82750000  1.106937
##    chr17.chr17_82750001_83000000  0.818990
##    chr17.chr17_83000001_83250000  0.874038
##    chr17.chr17_83250001_83257441  0.000000
##    -------
##    seqinfo: 1 sequence from an unspecified genome
```

7.1.3 Exporting compartment tracks

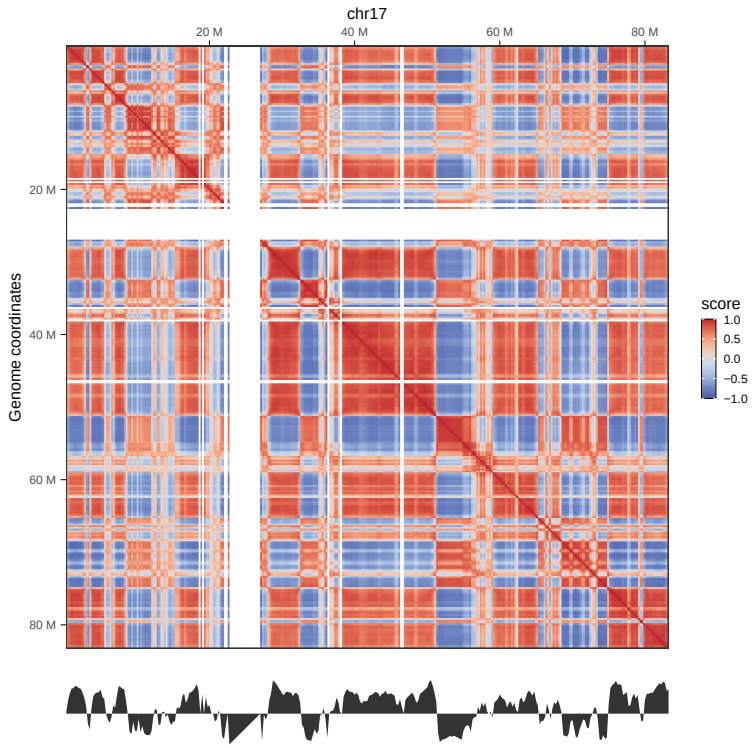
To save the eigenvector (as a `bigwig` file) and the compartments(as a `gff` file), the `export` function can be used:

```
library(GenomicRanges)
library(rtracklayer)
coverage(metadata(microC_compts)$eigens, weight = 'eigen') |> export('microC_eigen.bw')
topologicalFeatures(microC_compts, "compartments") |> export('microC_compartments.gff3')
```

7.1.4 Visualizing compartment tracks

Compartment tracks should be visualized in a dedicated genome browser, with the phasing track loaded as well, to ensure they are phased accordingly.
That being said, it is possible to visualize a genome track in R besides the matching Hi-C contact matrix.

```
library(ggplot2)
library(patchwork)
microC <- autocorrelate(microC)
##
p1 <- plotMatrix(microC, use.scores = 'autocorrelated', scale = 'linear', limits = c(-1, 1),
eigen <- coverage(metadata(microC_compts)$eigens, weight = 'eigen')[[1]]
eigen_df <- tibble(pos = cumsum(runLength(eigen)), eigen = runValue(eigen))
p2 <- ggplot(eigen_df, aes(x = pos, y = eigen)) +
    geom_area() +
    theme_void() +
    coord_cartesian(expand = FALSE) +
    labs(x = "Genomic position", y = "Eigenvector value")
wrap_plots(p1, p2, ncol = 1, heights = c(10, 1))
```



Here, we clearly note the concordance between the Hi-C correlation matrix, highlighting correlated interactions between pairs

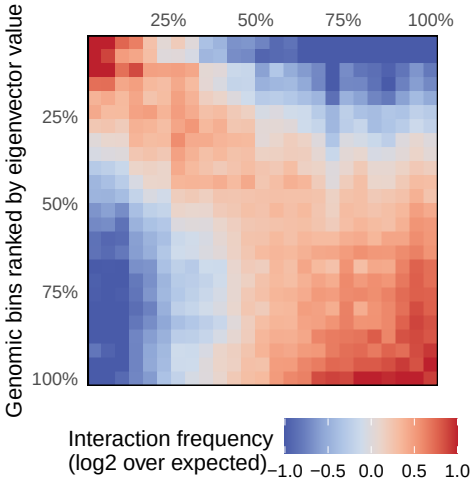
of genomic segments, and the eigenvector representing chromosome segmentation into 2 compartments: A (for positive values) and B (for negative values).

7.1.5 Saddle plots

Saddle plots are typically used to measure the `observed` vs. `expected` interaction scores within or between genomic loci belonging to A and B compartments.

Non-overlapping genomic windows are grouped in `nbins` quantiles (typically between 10 and 50 quantiles) according to their A/B compartment eigenvector value, from lowest eigenvector values (i.e. strongest B compartments) to highest eigenvector values (i.e. strongest A compartments). The average `observed` vs. `expected` interaction scores are then computed for pairwise eigenvector quantiles and plotted in a 2D heatmap.

```
library(BiocParallel)
plotSaddle(microC_compts, nbins = 25, BPPARAM = SerialParam(progressbar = FALSE))
```



Here, the top-left small corner represents average O/E scores between strong B compartments and the bottom-right larger

corner represents average O/E scores between strong A compartments. Note that only `chr17` interactions are contained in this dataset, explaining the grainy aspect of the saddle plot.

7.2 Topological domains

Topological *domains* (a.k.a. Topologically Associating Domains, TADs, isolated neighborhoods, contact domains, …) refer to local chromosomal segments (e.b. roughly 1Mb in mammal genomes) which preferentially self-interact, in a constrained manner. They are demarcated by domain *boundaries*.

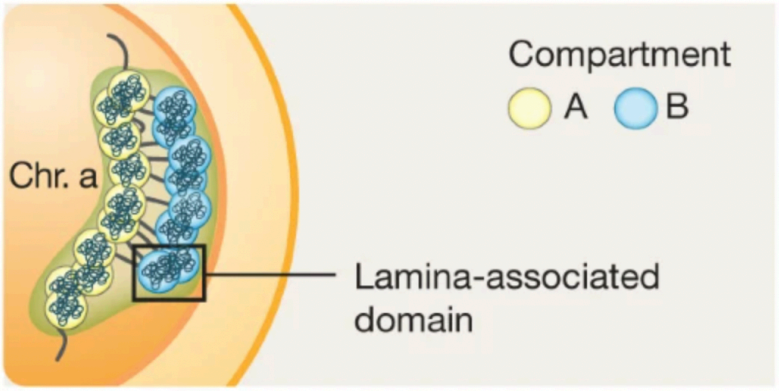
They are generally conserved across cell types and species (@Schmitt_2016), typically correlate with units of DNA replication (@Pope_2014), and could play a role during development (@Stadhouders_2019).

7.2.1 Computing diamond insulation score

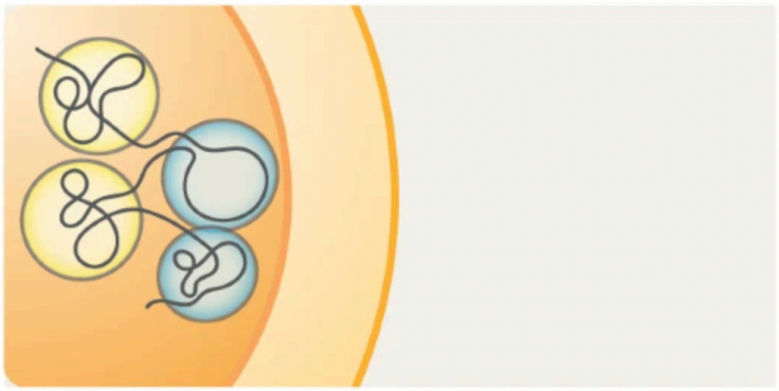
Several approaches exist to annotate topological domains (@Sefer_2022). Several packages in R implement some of these functionalities, e.g. `spectralTAD` or `TADcompare`.

`HiContacts` offers a simple `getDiamondInsulation` function which computes the diamond insulation score (@Crane_2015). This score quantifies average interaction frequency in an insulation window (of a certain `window_size`) sliding along contact matrices at a chosen `resolution`.

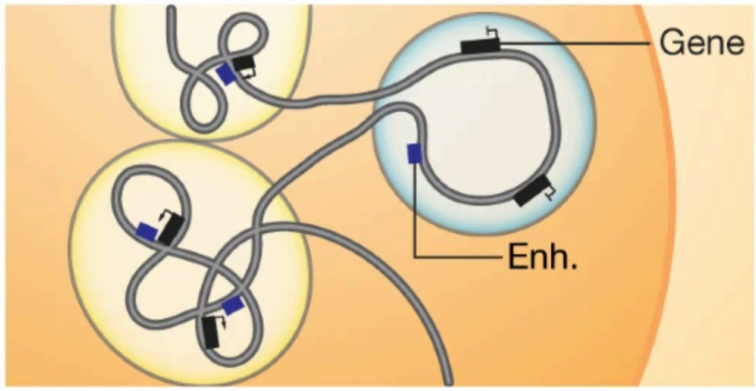
```
# - Compute insulation score
bpparam <- SerialParam(progressbar = FALSE)
hic <- zoom(microC, 5000) |>
    refocus('chr17:60000001-83257441') |>
    getDiamondInsulation(window_size = 100000, BPPARAM = bpparam) |>
    getBorders()
##  Going through preflight checklist...
##  Scan each window and compute diamond insulation score...
##  Annotating diamond score prominence for each window...
```



TADs and loop domains



Intra-TAD interactions



```
hic
##  `HiCExperiment` object with 2,156,222 contacts over 4,652 regions
##  -------
##  fileName: "/tmp/RtmpEEUcH8/Rinst555f2f89bc/OHCA/extdata/chr17.mcool"
##  focus: "chr17:60,000,001-83,257,441"
##  resolutions(3): 5000 100000 250000
##  active resolution: 5000
##  interactions: 2156044
##  scores(2): count balanced
##  topologicalFeatures: compartments(0) borders(21) loops(0) viewpoints(0)
##  pairsFile: N/A
##  metadata(1): insulation
```

`getDiamondInsulation()` is an endomorphism: it returns the original object, enriched with two new pieces of information:

- A `borders` `topologicalFeatures`:

```
topologicalFeatures(hic, "borders")
##  GRanges object with 21 ranges and 1 metadata column:
##           seqnames            ranges strand |     score
##              <Rle>         <IRanges>  <Rle> | <numeric>
##    strong    chr17 60105001-60110000      * |  0.574760
##      weak    chr17 60210001-60215000      * |  0.414425
##      weak    chr17 61415001-61420000      * |  0.346668
##    strong    chr17 61500001-61505000      * |  0.544336
##      weak    chr17 62930001-62935000      * |  0.399794
##       ...      ...                ...    ... .       ...
##      weak    chr17 78395001-78400000      * |  0.235613
##      weak    chr17 79065001-79070000      * |  0.236535
##      weak    chr17 80155001-80160000      * |  0.284855
##      weak    chr17 81735001-81740000      * |  0.497478
##    strong    chr17 81840001-81845000      * |  1.395949
##    -------
##    seqinfo: 1 sequence from an unspecified genome
```

- The calculated `insulation` scores stored in `metadata`:

```
metadata(hic)$insulation
##  GRanges object with 4611 ranges and 8 metadata columns:
##                             seqnames            ranges strand |    bin_id
##                                <Rle>         <IRanges>  <Rle> | <numeric>
##    chr17_60100001_60105000    chr17 60100001-60105000      * |     12020
##    chr17_60105001_60110000    chr17 60105001-60110000      * |     12021
##    chr17_60110001_60115000    chr17 60110001-60115000      * |     12022
##    chr17_60115001_60120000    chr17 60115001-60120000      * |     12023
##    chr17_60120001_60125000    chr17 60120001-60125000      * |     12024
##                        ...      ...                 ...    ... .       ...
##    chr17_83130001_83135000    chr17 83130001-83135000      * |     16626
##    chr17_83135001_83140000    chr17 83135001-83140000      * |     16627
##    chr17_83140001_83145000    chr17 83140001-83145000      * |     16628
##    chr17_83145001_83150000    chr17 83145001-83150000      * |     16629
##    chr17_83150001_83155000    chr17 83150001-83155000      * |     16630
##                               weight   chr    center     score insulation
##                            <numeric> <Rle> <integer> <numeric>  <numeric>
##    chr17_60100001_60105000 0.0406489 chr17  60102500  0.188061  -0.750142
##    chr17_60105001_60110000 0.0255539 chr17  60107500  0.180860  -0.806466
##    chr17_60110001_60115000       NaN chr17  60112500  0.196579  -0.686232
##    chr17_60115001_60120000       NaN chr17  60117500  0.216039  -0.550046
##    chr17_60120001_60125000       NaN chr17  60122500  0.230035  -0.459489
##                        ...       ...   ...       ...       ...        ...
##    chr17_83130001_83135000 0.0314684 chr17  83132500  0.262191  -0.270723
##    chr17_83135001_83140000 0.0307197 chr17  83137500  0.240779  -0.393632
##    chr17_83140001_83145000 0.0322810 chr17  83142500  0.219113  -0.529664
##    chr17_83145001_83150000 0.0280840 chr17  83147500  0.199645  -0.663900
##    chr17_83150001_83155000 0.0272775 chr17  83152500  0.180434  -0.809873
##                                  min prominence
##                            <logical>  <numeric>
##    chr17_60100001_60105000     FALSE         NA
##    chr17_60105001_60110000      TRUE    0.57476
##    chr17_60110001_60115000     FALSE         NA
##    chr17_60115001_60120000     FALSE         NA
##    chr17_60120001_60125000     FALSE         NA
##                        ...       ...        ...
##    chr17_83130001_83135000     FALSE         NA
##    chr17_83135001_83140000     FALSE         NA
##    chr17_83140001_83145000     FALSE         NA
```

```
##     chr17_83145001_83150000      FALSE          NA
##     chr17_83150001_83155000      FALSE          NA
##     -------
##     seqinfo: 1 sequence from an unspecified genome
```

> **Note**
>
> The `getDiamondInsulation` function can be parallelized over multiple threads by specifying the Bioconductor generic `BPPARAM` argument.

7.2.2 Exporting insulation scores tracks

To save the diamond insulation scores (as a `bigwig` file) and the borders (as a `bed` file), the `export` function can be used:

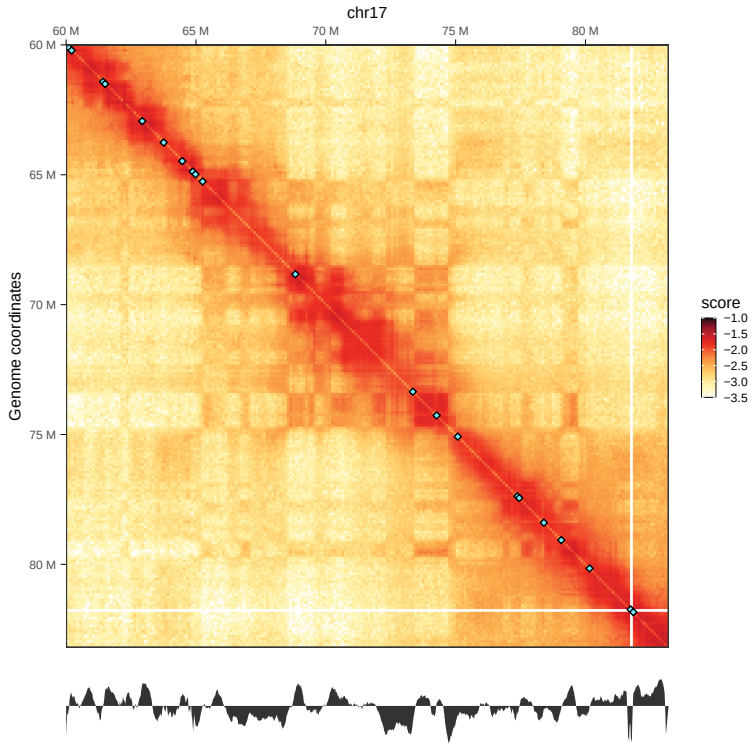
```
coverage(metadata(hic)$insulation, weight = 'insulation') |> export('microC_insulation.bw')
topologicalFeatures(hic, "borders") |> export('microC_borders.bed')
```

7.2.3 Visualizing chromatin domains

Insulation tracks should be visualized in a dedicated genome browser.
That being said, it is possible to visualize a genome track in R besides the matching Hi-C contact matrix.

```
hic <- zoom(hic, 100000)
p1 <- plotMatrix(
    hic,
    use.scores = 'balanced',
    limits = c(-3.5, -1),
    borders = topologicalFeatures(hic, "borders"),
    caption = FALSE
)
insulation <- coverage(metadata(hic)$insulation, weight = 'insulation')[[1]]
insulation_df <- tibble(pos = cumsum(runLength(insulation)), insulation = runValue(insulatio
p2 <- ggplot(insulation_df, aes(x = pos, y = insulation)) +
```

```
    geom_area() +
    theme_void() +
    coord_cartesian(expand = FALSE) +
    labs(x = "Genomic position", y = "Diamond insulation score")
wrap_plots(p1, p2, ncol = 1, heights = c(10, 1))
```



Local minima in the diamond insulation score displayed below the Hi-C contact matrix are identified using the `getBorders()` function, which automatically estimates a minimum threshold. These local minima correspond to borders and are visually depicted on the Hi-C map by blue diamonds.

7.3 Chromatin loops

7.3.1 `chromosight`

Chromatin loops, dots, or contacts, refer to a strong increase of interaction frequency between a pair of two genomic loci. They correspond to focal "dots" on a Hi-C map. Relying on computer vision algorithms, `chromosight` uses this property to annotate chromatin loops in a Hi-C map (@Matthey_Doret_2020). `chromosight` is a standalone `python` package and is made available in R through the `HiCool`-managed conda environment with the `getLoops()` function.

7.3.1.1 Identifying loops

```
hic <- HiCool::getLoops(microC, resolution = 5000)

hic
## `HiCExperiment` object with 917,156 contacts over 100 regions
## -------
## fileName: "/home/rsg/.cache/R/fourDNData/4d434d8538a0_4DNFI9FVHJZQ.mcool"
## focus: "chr17:63,000,001-63,500,000"
## resolutions(13): 1000 2000 ... 5000000 10000000
## active resolution: 5000
## interactions: 5047
## scores(2): count balanced
## topologicalFeatures: compartments(0) borders(0) loops(66411) viewpoints(0)
## pairsFile: N/A
## metadata(1): chromosight_args
```

`getLoops()` is an endomorphism: it returns the original object, enriched with two new pieces of information:

- A `loops` `topologicalFeatures`:

```
topologicalFeatures(hic, "loops")
## GInteractions object with 66411 interactions and 5 metadata columns:
##          seqnames1           ranges1     seqnames2           ranges2 |   bin_id1   bin_i
##              <Rle>         <IRanges>         <Rle>         <IRanges> | <numeric> <numeri
```

```
##       [1]      chr1     775001-780000 ---      chr1     850001-855000 |       155       1
##       [2]      chr1     775001-780000 ---      chr1     865001-870000 |       155       1
##       [3]      chr1     865001-870000 ---      chr1     890001-895000 |       173       1
##       [4]      chr1     910001-915000 ---      chr1     955001-960000 |       182       1
##       [5]      chr1     910001-915000 ---      chr1   1055001-1060000 |       182       2
##       ...       ...                ... ...       ...                ... .       ...       .
##   [66407]      chrY 19570001-19575000 ---      chrY 19720001-19725000 |    610133    6101
##   [66408]      chrY 19705001-19710000 ---      chrY 19730001-19735000 |    610160    6101
##   [66409]      chrY 19765001-19770000 ---      chrY 19800001-19805000 |    610172    6101
##   [66410]      chrY 20555001-20560000 ---      chrY 20645001-20650000 |    610330    6103
##   [66411]      chrY 21015001-21020000 ---      chrY 21055001-21060000 |    610422    6104
##   -------
##   regions: 84171 ranges and 0 metadata columns
##   seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

- The arguments used by `chromosight`, stored in `metadata`:

```
metadata(hic)$chromosight_args
## $`--pattern`
## [1] "loops"
##
## $`--dump`
## [1] "/data/.cache/R//RtmpSaRwiZ"
##
## $`--inter`
## [1] FALSE
##
## $`--iterations`
## [1] "auto"
##
## $`--kernel-config`
## NULL
##
## $`--perc-zero`
## [1] "auto"
##
## $`--perc-undetected`
## [1] "auto"
```

```
##
## $`--tsvd`
## [1] FALSE
##
## $`--win-fmt`
## [1] "json"
##
## $`--win-size`
## [1] "auto"
##
## $`--no-plotting`
## [1] TRUE
##
## $`--smooth-trend`
## [1] FALSE
##
## $`--norm`
## [1] "auto"
##
## $`<contact_map>`
## [1] "/home/rsg/.cache/R/fourDNData/4d434d8538a0_4DNFI9FVHJZQ.mcool::/resolutions/5000"
##
## $`--max-dist`
## [1] "auto"
##
## $`--min-dist`
## [1] "auto"
##
## $`--min-separation`
## [1] "auto"
##
## $`--n-mads`
## [1] 5
##
## $`<prefix>`
## [1] "chromosight/chromo"
##
## $`--pearson`
## [1] "auto"
##
```

```
## $`--subsample`
## [1] "no"
##
## $`--threads`
## [1] 1
```

7.3.1.2 Importing loops from files

If you are using `chromosight` directly from the terminal (i.e. outside `R`), you can import the annotated loops in `R` as follows:

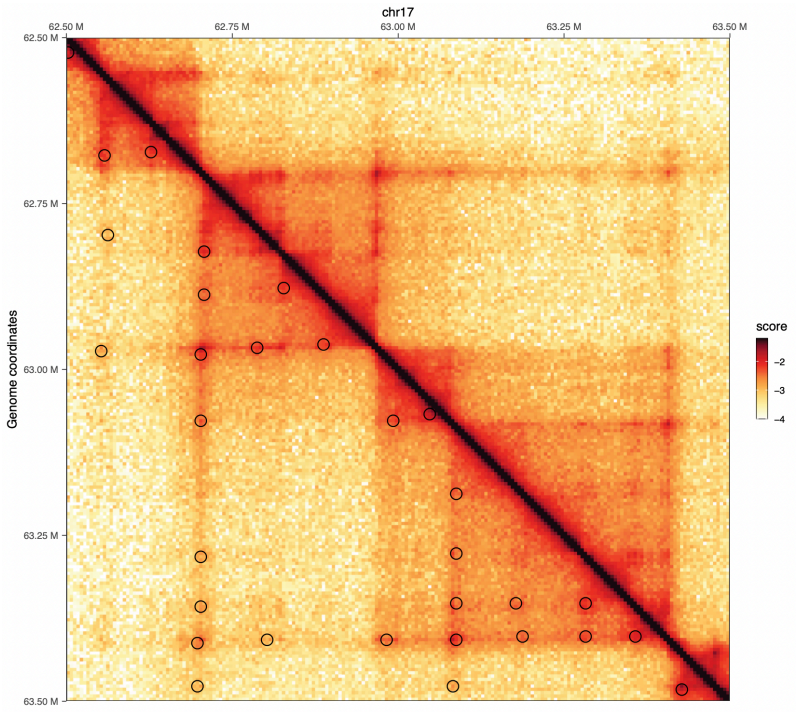
```
df <- readr::read_tsv("...") ## Here put your loops file
loops <- InteractionSet::GInteractions(
    anchor1 = GenomicRanges::GRanges(
        df$chrom1, IRanges::IRanges(df$start1+1, df$end1)
    ),
    anchor2 = GenomicRanges::GRanges(
        df$chrom2, IRanges::IRanges(df$start2+1, df$end2)
    ),
    bin_id1 = df$bin1,
    bin_id2 = df$bin2,
    score = df$score,
    pvalue = df$pvalue,
    qvalue = df$qvalue
)
```

7.3.1.3 Exporting chromatin loops

```
loops <- topologicalFeatures(hic, "loops")
loops <- loops[loops$score >= 0.4 & loops$qvalue <= 1e-6]
GenomicInteractions::export.bedpe(loops, 'loops.bedpe')
```

7.3.1.4 Visualizing chromatin loops

```
plotMatrix(
    refocus(hic, 'chr17:62500001-63500000') |> zoom(5000),
    loops = loops,
    limits = c(-4, -1.2),
    caption = FALSE
)
```



7.3.2 Other R packages

A number of other R packages have been developed to identify focal chromatin loops, notably `fitHiC` (@Ay_2014), `GOTHiC` (@Mifsud_2017) or `idr2d` (@Krismer_2020). Each fits a slightly different purpose, and we encourage the end user to read companion publications.

References

Part III

Advanced Hi-C topics

8 Data gateways: accessing public Hi-C data portals

Aims

This chapter focuses on introducing two important portals hosting public Hi-C datasets: the 4DN Consortium and the DNA Zoo project. Two `R` packages provide a programmatic access to these portals:

- `fourDNData`
- `DNAZooData`

The Hi-C experimental approach has gained significant traction across multiple fields related to genome biology, and several consortia developed large-scale programs based on this technique. The `fourDNData` and `DNAZooData` R packages were designed to accelerate the investigation of chromatin structure using these public resources.

8.1 4DN data portal

The 4D Nucleome Data Coordination and Integration Center (DCIC) has developed and actively maintains a data portal providing public access to a wealth of resources to investigate 3D chromatin architecture. Notably, 3D chromatin conformation libraries relying on different technologies ("in situ" or "dilution" Hi-C, Capture Hi-C, Micro-C, DNase Hi-C, ...), generated by 50+ collaborating labs, were homogenously processed, yielding more than 350 sets of processed files.

`fourDNData` (read *4DN-Data*) is a package giving programmatic access to these uniformly processed Hi-C contact files.

The `fourDNData()` function provides a gateway to 4DN-hosted Hi-C files, including contact matrices (in `.hic` or `.mcool`) and other Hi-C derived files such as annotated compartments, domains, insulation scores, or `.pairs` files.

```
library(fourDNData)
head(fourDNData())
##  experimentSetAccession     fileType     size organism experimentType details
##            4DNES18BMU79        pairs 10151.53    mouse   in situ Hi-C   DpnII Hi-C on Mou
##            4DNES18BMU79          hic  5285.82    mouse   in situ Hi-C   DpnII Hi-C on Mou
##            4DNES18BMU79        mcool  6110.75    mouse   in situ Hi-C   DpnII Hi-C on Mou
##            4DNES18BMU79   boundaries     0.12    mouse   in situ Hi-C   DpnII Hi-C on Mou
##            4DNES18BMU79   insulation     7.18    mouse   in situ Hi-C   DpnII Hi-C on Mou
##            4DNES18BMU79 compartments     0.18    mouse   in situ Hi-C   DpnII Hi-C on Mou
```

8.1.1 Querying individual files

The `fourDNData()` function can be used to directly fetch specific files from the 4DN data portal:

```
cf <- fourDNData(experimentSetAccession = '4DNES25ABNZ1', type = 'mcool')
##  |==================================|  100%
```

This effectively downloads and caches the queried file locally.

```
cf
## [1] "/home/rsg/.cache/R/fourDNData/698470d302f0_4DNFI4988896.mcool"

availableChromosomes(cf)
## Seqinfo object with 24 sequences from an unspecified genome:
##   seqnames seqlengths isCircular genome
##   chr1      248956422       <NA>   <NA>
##   chr2      242193529       <NA>   <NA>
##   chr3      198295559       <NA>   <NA>
##   chr4      190214555       <NA>   <NA>
##   chr5      181538259       <NA>   <NA>
##   ...             ...        ...    ...
##   chr20      64444167       <NA>   <NA>
##   chr21      46709983       <NA>   <NA>
```

```
##   chr22      50818468       <NA>   <NA>
##   chrX      156040895       <NA>   <NA>
##   chrY       57227415       <NA>   <NA>

availableResolutions(cf)
## resolutions(13): 1000 2000 ... 5000000 10000000

import(cf, focus = "chr4:10000001-20000000", resolution = 5000)
## `HiCExperiment` object with 14,682 contacts over 2,000 regions
## -------
## fileName: "/home/rsg/.cache/R/fourDNData/29051ff3104c_4DNFINSF15ZM.mcool"
## focus: "chr4:10,000,001-20,000,000"
## resolutions(13): 1000 2000 ... 5000000 10000000
## active resolution: 5000
## interactions: 12016
## scores(2): count balanced
## topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
## pairsFile: N/A
## metadata(0):
```

Different Hi-C related genomic files are provided by the 4DN consortium. The type of file to fetch can be specified with the `type` argument:

- `type = 'pairs'` will fetch the pairs file which was generated by the 4DN consortium and binned into a contact matrix. Once fetched from the 4DN data portal, the local file can be imported in `R` using the `import` function, which will generate a `GInteractions` object.

```
fourDNData(experimentSetAccession = '4DNES25ABNZ1', type = 'pairs') |>
    import()
## GInteractions object with 13821669 interactions and 3 metadata columns:
##                seqnames1   ranges1     seqnames2   ranges2 |       frag1     frag2  distanc
##                    <Rle> <IRanges>         <Rle> <IRanges> | <character> <numeric> <integer
##          [1]      chr1   3000003 ---      chr1  88307603 |          UU        22  8530760
##          [2]      chr1   3000022 ---      chr1  28227919 |          UU        50  2522789
##          [3]      chr1   3000023 ---      chr1  50187758 |          RU        35  4718773
##          [4]      chr1   3000024 ---      chr1   4090828 |          RU         9   109080
##          [5]      chr1   3000024 ---      chr1  35080614 |          UU         3  3208059
```

```
##          ...       ...       ... ...       ...       ... .          ...       ...       ..
##   [13821665]      chr1  24472292 ---      chr1  24472986 |           UU        60        69
##   [13821666]      chr1  24472292 ---      chr1  24805552 |           RU        60     33326
##   [13821667]      chr1  24472292 ---      chr1  24874144 |           UU        60     40185
##   [13821668]      chr1  24472294 ---      chr1 115668400 |           UU        60    911961
##   [13821669]      chr1  24472295 ---      chr1  43307467 |           UU        60    188351
```

> ⚠ Watch out
>
> `.pairs` files can be particularly large and therefore will take both and long time to download and a large storage footprint.

- `type = 'insulation'` will fetch a `.bigwig` track file precomputed by the 4DN consortium. This track corresponds to the genome-wide insulation score computed by `cooltools` as described in @Crane_2015. To know more about this, read the excerpt from 4DN data portal. Once fetched from the 4DN data portal, the local file can be imported in `R` using the `import` function, which will generate a `RleList` object.

```
fourDNData(experimentSetAccession = '4DNES25ABNZ1', type = 'insulation') |>
    import(as = 'Rle')
##  |==================================|  100%
##  RleList of length 21
##  $chr1
##  numeric-Rle of length 195471971 with 38145 runs
##    Lengths:      3065000         5000 ...         5000       171971
##    Values :  0.00000e+00  1.01441e-01 ...     0.807009     0.000000
##
##  $chr10
##  numeric-Rle of length 130694993 with 25100 runs
##    Lengths:     3175000        5000        5000 ...        5000      169993
##    Values :  0.00000000  0.37584546  0.33597839 ...    0.628601    0.000000
##
##  $chr11
##  numeric-Rle of length 122082543 with 23536 runs
##    Lengths:    3165000       5000       5000 ...       5000     162543
```

```
##    Values :  0.0000000 -0.7906257 -0.7930040 ...   0.515919   0.000000
##
##  ...
```

- `type = 'boundaries'` will fetch a `.bed` file precomputed by the 4DN consortium, listing the annotated borders between topological domains. These borders correspond to local minima identified from the genome-wide insulation track. It can also be imported in `R` using the `import` function, which will generate a `GRanges` object.

```
fourDNData(experimentSetAccession = '4DNES25ABNZ1', type = 'boundaries') |>
    import()
##  |==================================|  100%
##  GRanges object with 6103 ranges and 2 metadata columns:
##           seqnames            ranges strand |        name     score
##              <Rle>         <IRanges>  <Rle> | <character> <numeric>
##       [1]     chr1   4380001-4385000      * |      Strong  0.695274
##       [2]     chr1   4760001-4765000      * |        Weak  0.444476
##       [3]     chr1   4910001-4915000      * |        Weak  0.353184
##       [4]     chr1   5180001-5185000      * |      Strong  0.565763
##       [5]     chr1   6170001-6175000      * |      Strong  1.644911
##       ...      ...               ...    ... .         ...       ...
##    [6099]     chrY 89725001-89730000      * |        Weak  0.258094
##    [6100]     chrY 89790001-89795000      * |        Weak  0.442186
##    [6101]     chrY 89895001-89900000      * |        Weak  0.279879
##    [6102]     chrY 90025001-90030000      * |      Strong  0.660699
##    [6103]     chrY 90410001-90415000      * |      Strong  1.160018
```

- `type = 'compartments'` will fetch a `.bigwig` track file precomputed by the 4DN consortium. This track corresponds to the selected genome-wide eigenvector computed by `cooltools` and representing A/B compartments. To know more about this, read the excerpt from 4DN data portal. Once fetched from the 4DN data portal, the local file can be imported in `R` using the `import` function, which will generate a `RleList` object. The score represents the eigenvector values, and by convention a genomic bin with a positive score is associated with the A compartment whereas a genomic bin with a negative score is associated

with the B compartment.

```
fourDNData(experimentSetAccession = '4DNES25ABNZ1', type = 'compartments') |>
    import()
##  |==================================|  100%
##  RleList of length 21
##  $chr1
##  numeric-Rle of length 195471971 with 771 runs
##    Lengths:     3000000      250000      250000 ...      250000      221971
##    Values :         NaN -0.83457172 -0.98202854 ...  0.45792237         NaN
##
##  $chr10
##  numeric-Rle of length 130694993 with 512 runs
##    Lengths:     3000000      250000      250000 ...      250000      194993
##    Values :         NaN -0.99524581 -0.76405841 ...   0.0583894         NaN
##
##  $chr11
##  numeric-Rle of length 122082543 with 478 runs
##    Lengths:     3000000      250000      250000 ...      250000       82543
##    Values :         NaN -0.00653325  0.26659977 ...  0.25900587         NaN
```

8.1.2 Querying a complete experiment dataset

Rather than importing multiple files corresponding to a single experimentSet accession ID one by one, one can import all the available files associated with a experimentSet accession ID into a `HiCExperiment` object by using the `fourDNHiCExperiment()` function.

```
hic <- fourDNHiCExperiment('4DNESSS7VU57')
## Fetching Hi-C contact map from 4DN portal
##   |======================================================================| 100%
##
## Compartments not found for the provided experimentSet accession.
## Fetching insulation bigwig file from 4DN portal
##   |======================================================================| 100%
##
## Fetching borders bed file from 4DN portal
##   |======================================================================| 100%
```

This is a more efficient way to import datasets, as it aggregates the different bits together into a single `HiCExperiment` object with populated `topologicalFeatures` and `metadata` slots.

```
hic
## `HiCExperiment` object with 544,370,135 contacts over 286 regions
## -------
## fileName: "/home/rsg/.cache/R/fourDNData/392eba3a587_4DNFIZ59STGB.mcool"
## focus: "whole genome"
## resolutions(13): 1000 2000 ... 5000000 10000000
## active resolution: 10000000
## interactions: 40088
## scores(2): count balanced
## topologicalFeatures: compartments(0) borders(7887)
## pairsFile: N/A
## metadata(2): 4DN_info diamond_insulation
```

```
metadata(hic)
##  $`4DN_info`
##  experimentSetAccession   fileType    size organism experimentType        details
##            4DNESSS7VU57      pairs 9731.58    mouse   in situ Hi-C Arima - A1, A2 Hi-C on
##            4DNESSS7VU57        hic 4160.17    mouse   in situ Hi-C Arima - A1, A2 Hi-C on
##            4DNESSS7VU57      mcool 2863.90    mouse   in situ Hi-C Arima - A1, A2 Hi-C on
##            4DNESSS7VU57 insulation    7.25    mouse   in situ Hi-C Arima - A1, A2 Hi-C on
##            4DNESSS7VU57 boundaries    0.12    mouse   in situ Hi-C Arima - A1, A2 Hi-C on
##
##  $diamond_insulation
##  RleList of length 20
##      $chr1
##      numeric-Rle of length 195471971 with 37959 runs
##        Lengths:    3085000       5000       5000 ...       5000     186971
##        Values :  0.0000000  0.3967191  0.3961740 ...  0.8223819  0.0000000
##
##      $chr10
##      numeric-Rle of length 130694993 with 24994 runs
##        Lengths:      3180000         5000 ...         5000       179993
##        Values :  0.00000e+00  5.35871e-01 ...   0.60626638   0.00000000
##
##      ...
```

8.2 DNA Zoo

The DNA Zoo Consortium is a collaborative group whose aim is to correct and refine genome assemblies across the tree of life using Hi-C approaches. As of 2023, they have performed Hi-C across more than 300 animal, plant and fungi species.

`DNAZooData` is a package giving programmatic access to these uniformly processed Hi-C contact files, as well as the refined genome assemblies.

The `DNAZooData()` function provides a gateway to DNA Zoo-hosted Hi-C files, fetching and caching relevant contact matrices in `.hic` format It returns a `HicFile` object, which can then be imported in memory using `import()`.

```
library(DNAZooData)
head(DNAZooData())
##                  species                                 readme
##         Acinonyx_jubatus        Acinonyx_jubatus/README.json        https://dnazoo.s3.was
##       Acropora_millepora      Acropora_millepora/README.json      https://dnazoo.s3.wasab
##      Addax_nasomaculatus     Addax_nasomaculatus/README.json     https://dnazoo.s3.wasabi
##            Aedes_aegypti           Aedes_aegypti/README.json           https://dnazoo.s3.
##    Aedes_aegypti__AaegL4   Aedes_aegypti__AaegL4/README.json   https://dnazoo.s3.wasabisy
##  Aedes_aegypti__AaegL5.0 Aedes_aegypti__AaegL5.0/README.json https://dnazoo.s3.wasabisys.
```

For example, we can directly fetch a Hi-C dataset generated from a tardigrade sample by specifying the right `species` argument.

```
hicfile <- DNAZooData(species = 'Hypsibius_dujardini')
##  Fetching Hi-C data from DNAZoo
##  |==================================|  100%
hicfile
##  HicFile object
##  .hic file: /home/rsg/.cache/R/DNAZooData/400d7e2b0145_nHd_3.1_HiC.hic
##  resolution: 5000
##  pairs file:
##  metadata(6): organism draftAssembly ... credits assemblyURL
```

Here again, the resulting `HicFile` is populated with metadata parsed from the DNA Zoo data portal.

```
metadata(hicfile)$organism
##  $vernacular
##  [1] "Tardigrade"
##
##  $binomial
##  [1] "Hypsibius dujardini"
##
##  $funFact
##  [1] "<i>Hypsibius dujardini</i> is a species of tardigrade, a tiny microscopic organism.
##
##  $extraInfo
##  [1] "on BioKIDS website"
##
##  $extraInfoLink
##  [1] "http://www.biokids.umich.edu/critters/Hypsibius_dujardini/"
##
##  $image
##  [1] "https://static.wixstatic.com/media/2b9330_82db39c219f24b20a75cb38943aad1fb~mv2.jpg"
##
##  $imageCredit
##  [1] "By Willow Gabriel, Goldstein Lab - https://www.flickr.com/photos/waterbears/161409
##  d=2261992"
##
##  $isChromognomes
##  [1] "FALSE"
##
##  $taxonomy
##  [1] "Species:202423-914154-914155-914158-155166-155362-710171-710179-710192-155390-15542
```

`HiCFile` metadata also points to a URL to directly fetch the genome assembly corrected by the DNA Zoo consortium.

```
metadata(hicfile)$assemblyURL
##  [1] "https://dnazoo.s3.wasabisys.com/Hypsibius_dujardini/nHd_3.1_HiC.fasta.gz"
```

References

9 Interoperability: using HiCExperiment with other R packages

> **Aims**
>
> This notebook illustrates how to use a range of popular Hi-C—related `R` packages with `HiCExperiment` objects. Conversion to the data structures supported by the following packages is illustrated here:
>
> - `diffHic`
> - `multiHiCcompare`
> - `TopDom`
> - `GOTHiC`

9.1 diffHic

`diffHic` is the first R package dedicated to Hi-C processing and analysis (@Lun_2015). It is packed with useful functions to generate a contact matrix from read pairs and to perform downstream investigation, including normalization, 2D "peak" (i.e. loops) finding and aggregation, differential interaction between samples, etc. It works seamlessly with the `InteractionSet` class of object, which can be easily obtained from a `HiCExperiment` object.

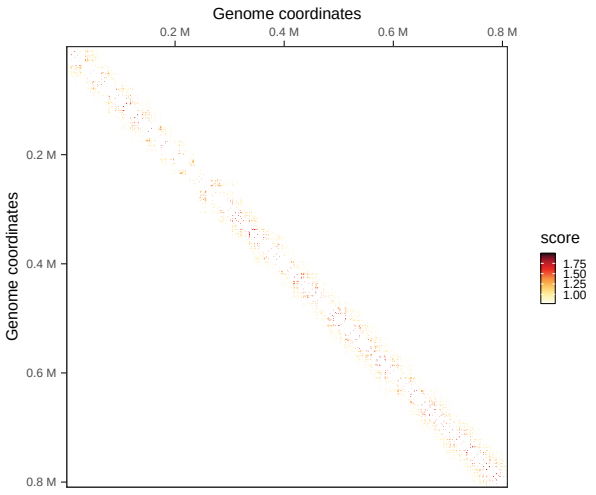
To do so, we first need to extract `GInteractions` from one or several `HiCExperiment` objects and **create** a single `InteractionSet` object.

```
library(InteractionSet)
library(GenomicRanges)
library(HiCExperiment)
library(HiContactsData)

# ---- This downloads an example `.mcool` file and caches it locally
coolf <- HiContactsData('yeast_wt', 'mcool')
##  see ?HiContactsData and browseVignettes('HiContactsData') for documentation
##  loading from cache
cool <- import(coolf, format = 'cool')
gi <- cool |>
    interactions() |>
    as("ReverseStrictGInteractions")
iset <- InteractionSet(
    assays = list(
        counts = matrix(gi$count, ncol = 1),
        balanced = matrix(gi$balanced, ncol = 1)
    ),
    interactions = gi,
    colData = data.frame(lib = c("WT"), totals = sum(gi$count))
)
```

From there, we can **filter** interactions to only retain those with significant enrichment over background.

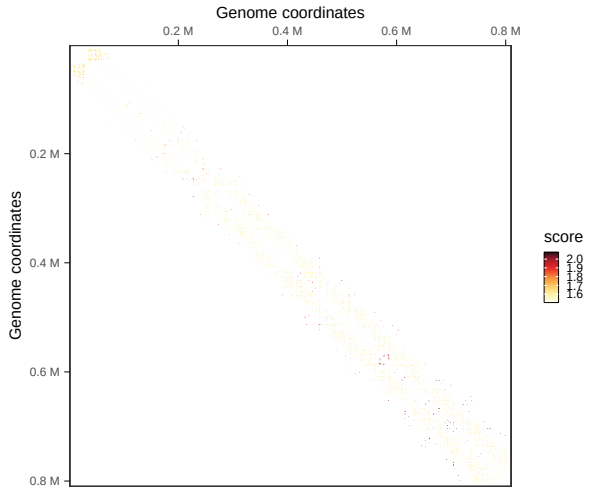
```
library(diffHic)
set.seed(1234)

# --- Filter to find aggregated interactions
enrichments <- enrichedPairs(iset)
filter <- filterPeaks(enrichments, min.enrich = log2(1.2), min.diag = 5)
filtered_iset <- iset[filter]
filtered_iset
##  class: InteractionSet
##  dim: 41872 1
##  metadata(0):
##  assays(2): counts balanced
##  rownames: NULL
##  rowData names(4): bin_id1 bin_id2 count balanced
##  colnames: NULL
```

```
##  colData names(2): lib totals
##  type: ReverseStrictGInteractions
##  regions: 12079

# --- Visualize filtered interactions
library(plyinteractions)
library(HiContacts)
##  Registered S3 methods overwritten by 'readr':
##    method                      from
##    as.data.frame.spec_tbl_df vroom
##    as_tibble.spec_tbl_df     vroom
##    format.col_spec           vroom
##    print.col_spec            vroom
##    print.collector           vroom
##    print.date_names          vroom
##    print.locale              vroom
##    str.col_spec              vroom
interactions(filtered_iset) |>
    filter(seqnames2 == 'II', seqnames1 == seqnames2) |>
    plotMatrix(use.scores = 'count')
```



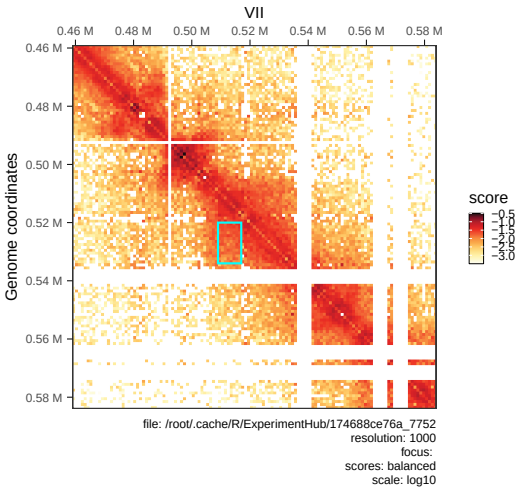
Next, we can cluster filtered interactions that are next to each other.

```
# --- Cluster interactions to find loops
clustered_iset <- clusterPairs(filtered_iset, tol = 5000)
clustered_iset$interactions
##  ReverseStrictGInteractions object with 1644 interactions and 0 metadata columns:
##           seqnames1       ranges1 strand1     seqnames2       ranges2 strand2
##               <Rle>     <IRanges>   <Rle>         <Rle>     <IRanges>   <Rle>
##       [1]         I  15001-149000       * ---         I      1-122000       *
##       [2]         I 133001-148000       * ---         I 127001-139000       *
##       [3]         I 154001-160000       * ---         I 128001-149000       *
##       [4]         I 168001-173000       * ---         I 138001-148000       *
##       [5]         I 184001-196000       * ---         I   15001-23000       *
##       ...       ...             ...     ... ...       ...             ...     ...
##    [1640]       XVI 897001-898000       * ---       XVI 831001-832000       *
##    [1641]       XVI 907001-910000       * ---       XVI 840001-843000       *
##    [1642]       XVI 926001-934000       * ---       XVI 872001-878000       *
##    [1643]       XVI 933001-934000       * ---       XVI 858001-859000       *
##    [1644]       XVI 933001-942000       * ---       XVI 928001-934000       *
##    -------
##    regions: 2822 ranges and 0 metadata columns
##    seqinfo: 16 sequences from an unspecified genome

# --- Visualize clustered interactions
interactions(filtered_iset) |>
    mutate(cluster = clustered_iset$indices[[1]]) |>
    filter(seqnames2 == 'II', seqnames1 == seqnames2) |>
    plotMatrix(use.scores = 'cluster')
```



Finally, we can visualize identified individual interaction clusters identified with `diffHic` using `HiContacts`.

```
# --- Plot matrix at a clustered loops
cgi <- clustered_iset$interactions[554]
seqn <- seqnames(anchors(cgi, type="second"))
start <- start(anchors(cgi, type="second")) - 50000
end <- end(anchors(cgi, type="first")) + 50000
interactions_peak <- GRanges(seqn, IRanges(start, end))
p <- plotMatrix(cool[interactions_peak])

library(ggplot2)
an <- anchors(cgi)
p + geom_rect(
    data = data.frame(xmin = start(an[[2]]), xmax = end(an[[2]]), ymin = start(an[[1]]), yma
    aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax),
    inherit.aes = FALSE,
    fill = NA,
    colour = 'cyan'
)
```



9.2 multiHiCcompare

The `multiHiCcompare` package provides functions for joint normalization and difference detection in multiple Hi-C datasets (@Stansfield_2019). According to its excerpt, to perform differential interaction analysis, it requires a `list` of **raw counts** for different samples/replicates, stored in **data frames with four columns** (`chr`, `start1`, `start2`, `count`).
Manipulate a `HiCExperiment` object to coerce it into such structure is straightforward.

```
library(dplyr)
library(tidyr)
library(purrr)
hics <- list(
    "wt" = import(coolf_wt, format = 'cool'),
    "eco1" = import(coolf_eco1, format = 'cool')
)
hics_list <- map(hics, ~ .x['XI'] |>
    as.data.frame() |>
    mutate(chr = 1) |>
    relocate(chr) |>
    select(chr, start1, start2, count)
```

```
)
head(hics_list[[1]])
##    chr start1 start2 count
## 1   1      1      1     2
## 2   1      1   1001     3
## 3   1      1   2001     3
## 4   1      1   3001    13
## 5   1      1   4001     9
## 6   1      1   5001    13
```

Once this list is generated, the classical `multiHiCcompare` workflow can be applied: first run `make_hicexp()`, followed by `cyclic_loess()`, then `hic_exactTest()` and finally `results()`:

```
DI <- hics_list |>
    make_hicexp(
        data_list = hics_list,
        groups = factor(c(1, 2))
    ) |>
    cyclic_loess() |>
    hic_exactTest() |>
    results()
DI
##         chr region1 region2 D      logFC    logCPM    p.value      p.adj
##      1:   1       1    1001 1  0.4279414  6.382927 0.78960192 1.0000000
##      2:   1       1    3001 3  1.0325237  8.339327 0.06035705 0.9501367
##      3:   1       1    4001 4  0.6862141  7.597689 0.34723639 1.0000000
##      4:   1       1    5001 5  0.5124878  7.960339 0.43133791 1.0000000
##      5:   1       1    6001 6 -0.3568672  8.563374 0.52289982 1.0000000
##     ---
##  22637:   1  663001  666001 3 -1.1680738  7.158551 0.17500113 1.0000000
##  22638:   1  664001  664001 0  1.4530501  8.536212 0.16535151 1.0000000
##  22639:   1  664001  665001 1 -0.1014769  8.166275 1.00000000 1.0000000
##  22640:   1  665001  665001 0 -0.3110054 10.013750 0.60075706 1.0000000
##  22641:   1  665001  666001 1 -0.4989794  7.750157 0.41481212 1.0000000
```

9.3 TopDom

The `TopDom` method is widely used to annotate topological domains in genomes from Hi-C data (@Shin_2015). The `TopDom` package was created to implement this method in `R` (@Bengtsson_2020).

Unfortunately, the format of the input to `TopDom` is rather tricky (see `?TopDom::readHiC`). The following chunk of code shows how to coerce a `HiCExperiment` object into a `TopDom`-compatible object.

```
library(TopDom)
hic <- import(coolf_wt, format = 'cool')
HiCExperiment2TopDom <- function(hic, chr) {
    data <- list()
    cm <- as(hic[chr], 'ContactMatrix')
    data$counts <- as.matrix(cm) |> base::as.matrix()
    data$counts[is.na(data$counts)] <- 0
    data$bins <- regions(cm) |>
        as.data.frame() |>
        select(seqnames, start, end) |>
        mutate(seqnames = as.character(seqnames)) |>
        mutate(id = 1:n(), start = start - 1) |>
        relocate(id) |>
        dplyr::rename(chr = seqnames, from.coord = start, to.coord = end)
    class(data) <- 'TopDomData'
    return(data)
}
hic_topdom <- HiCExperiment2TopDom(hic, "II")
hic_topdom
##  TopDomData:
##  bins:
##  'data.frame':   813 obs. of  4 variables:
##   $ id        : int  1 2 3 4 5 6 7 8 9 10 ...
##   $ chr       : chr  "II" "II" "II" "II" ...
##   $ from.coord: num  0 1000 2000 3000 4000 5000 6000 7000 8000 9000 ...
##   $ to.coord  : int  1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 ...
##  counts:
##   num [1:813, 1:813] 0 0 0 0 0 0 0 0 0 0 ...
```

Now that we have coerced a `HiCExperiment` object into a `TopDom`-compatible object, we can use the main `TopDom` function to annotate topological domains.

```
domains <- TopDom::TopDom(hic_topdom, window.size = 5)
domains
##  TopDom:
##  Parameters:
##  - window.size: 5
##  - statFilter: TRUE
##  binSignal:
##  'data.frame':   813 obs. of  7 variables:
##   $ id        : int  1 2 3 4 5 6 7 8 9 10 ...
##   $ chr       : chr  "II" "II" "II" "II" ...
##   $ from.coord: num  0 1000 2000 3000 4000 5000 6000 7000 8000 9000 ...
##   $ to.coord  : int  1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 ...
##   $ local.ext : num  -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 0 0 ...
##   $ mean.cf   : num  0 0 0 0 0 ...
##   $ pvalue    : num  1 1 1 1 1 ...
##  domain:
##  'data.frame':   61 obs. of  7 variables:
##   $ chr       : chr  "II" "II" "II" "II" ...
##   $ from.id   : int  1 9 31 36 47 61 76 82 91 102 ...
##   $ from.coord: num  0 8000 30000 35000 46000 60000 75000 81000 90000 101000 ...
##   $ to.id     : int  8 30 35 46 60 75 81 90 101 136 ...
##   $ to.coord  : num  8000 30000 35000 46000 60000 75000 81000 90000 101000 136000 ...
##   $ tag       : chr  "gap" "domain" "gap" "domain" ...
##   $ size      : num  8000 22000 5000 11000 14000 15000 6000 9000 11000 35000 ...
##  bed:
##  'data.frame':   61 obs. of  4 variables:
##   $ chrom     : chr  "II" "II" "II" "II" ...
##   $ chromStart: num  0 8000 30000 35000 46000 60000 75000 81000 90000 101000 ...
##   $ chromEnd  : num  8000 30000 35000 46000 60000 75000 81000 90000 101000 136000 ...
##   $ name      : chr  "gap" "domain" "gap" "domain" ...
```

The resulting `domains` object can be used to extract annotated domains, store them in `topologicalFeatures` of the original `HiCExperiment`, and optionally write a `bed` file to export them in text.

```
topologicalFeatures(hic, 'domain') <- domains$bed |>
    mutate(chromStart = chromStart + 1) |>
    filter(name == 'domain') |>
    makeGRangesFromDataFrame()
topologicalFeatures(hic, 'domain')
##  GRanges object with 52 ranges and 0 metadata columns:
##         seqnames        ranges strand
##            <Rle>     <IRanges>  <Rle>
##     [1]       II    8001-30000      *
##     [2]       II   35001-46000      *
##     [3]       II   46001-60000      *
##     [4]       II   60001-75000      *
##     [5]       II   75001-81000      *
##     ...      ...           ...    ...
##    [48]       II 664001-681000      *
##    [49]       II 681001-707000      *
##    [50]       II 707001-714000      *
##    [51]       II 714001-761000      *
##    [52]       II 761001-806000      *
##    -------
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths

rtracklayer::export(topologicalFeatures(hic, 'domain'), 'hic_domains.bed')
```

9.4 GOTHiC

`GOTHiC` relies on a cumulative binomial test to detect interactions between distal genomic loci that have significantly more reads than expected by chance in Hi-C experiments (@Mifsud_2017).

> **!** Using the `GOTHiC` function
>
> Unfortunately, the main `GOTHiC` function require two `.bam` files as input. These files are often deleted due to their larger size, while the filtered `pairs` file itself is retained. Moreover, the internal nuts and bolts of the main `GOTHiC` function perform several operations that are not required

in modern workflows:

1. Filtering pairs from same restriction fragment; this step is now usually taken care of automatically, e.g. with `HiCool` Hi-C processing package.
2. Filtering short-range pairs; the `GOTHiC` package hard-codes a 10kb lower threshold for minimum pair distance. More advanced optimized filtering approaches have been implemented since then, to circumvent the need for such hard-coded threshold.
3. Binning pairs; this step is also already taken care of, when working with Hi-C matrices in modern formats, e.g. with `.(m)cool` files.

Based on these facts, we can simplify the binomial test function provided by `GOTHiC` so that it can directly used binned interactions imported as a `HiCExperiment` object in `R`.

```
GOTHiC_binomial <- function(x) {

    if (length(trans(x)) != 0) stop("Only `cis` interactions can be used here.")
    ints <- interactions(x) |>
        as.data.frame() |>
        select(seqnames1, start1, seqnames2, start2, count) |>
        dplyr::rename(chr1 = seqnames1, locus1 = start1, chr2 = seqnames2, locus2 = start2,
        mutate(locus1 = locus1 - 1, locus2 = locus2 - 1) |>
        mutate(int1 = paste0(chr1, '_', locus1), int2 = paste0(chr2, '_', locus2))

    numberOfReadPairs <- sum(ints$frequencies)
    all_bins <- unique(c(unique(ints$int1), unique(ints$int2)))
    all_bins <- sort(all_bins)
    upperhalfBinNumber <- (length(all_bins)^2 - length(all_bins))/2

    cov <- ints |>
        group_by(int1) |>
        tally(frequencies) |>
        full_join(ints |>
            group_by(int2) |>
            tally(frequencies),
```

```
            by = c('int1' = 'int2')
        ) |>
        rowwise() |>
        mutate(coverage = sum(n.x, n.y, na.rm = TRUE)) |>
        ungroup() |>
        mutate(relative_coverage = coverage/sum(coverage))

    results <- mutate(ints,
        cov1 = left_join(ints, select(cov, int1, relative_coverage), by = c('int1' = 'int1'))
        cov2 = left_join(ints, select(cov, int1, relative_coverage), by = c('int2' = 'int1'))
        probability = cov1 * cov2 * 2 * 1/(1 - sum(cov$relative_coverage^2)),
        predicted = probability * numberOfReadPairs
    ) |>
    rowwise() |>
    mutate(
        pvalue = binom.test(
            frequencies,
            numberOfReadPairs,
            probability,
            alternative = "greater"
        )$p.value
    ) |>
    ungroup() |>
    mutate(
        logFoldChange = log2(frequencies / predicted),
        qvalue = stats::p.adjust(pvalue, method = "BH", n = upperhalfBinNumber)
    )

    scores(x, "probability") <- results$probability
    scores(x, "predicted") <- results$predicted
    scores(x, "pvalue") <- results$pvalue
    scores(x, "qvalue") <- results$qvalue
    scores(x, "logFoldChange") <- results$logFoldChange

    return(x)

}
```

```
res <- GOTHiC_binomial(hic["II"])
res
##  `HiCExperiment` object with 471,364 contacts over 802 regions
##  -------
##  fileName: "/root/.cache/R/ExperimentHub/174688ce76a_7752"
##  focus: "II"
##  resolutions(5): 1000 2000 4000 8000 16000
##  active resolution: 1000
##  interactions: 74360
##  scores(7): count balanced probability predicted pvalue qvalue logFoldChange
##  topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0) domain(52)
##  pairsFile: N/A
##  metadata(0):

interactions(res)
##  GInteractions object with 74360 interactions and 9 metadata columns:
##             seqnames1       ranges1 strand1     seqnames2       ranges2
##                 <Rle>     <IRanges>   <Rle>         <Rle>     <IRanges>
##        [1]        II        1-1000       * ---        II     1001-2000
##        [2]        II        1-1000       * ---        II     5001-6000
##        [3]        II        1-1000       * ---        II     6001-7000
##        [4]        II        1-1000       * ---        II     8001-9000
##        [5]        II        1-1000       * ---        II    9001-10000
##        ...       ...           ...     ... ...       ...           ...
##    [74356]        II 807001-808000       * ---        II 809001-810000
##    [74357]        II 807001-808000       * ---        II 810001-811000
##    [74358]        II 808001-809000       * ---        II 808001-809000
##    [74359]        II 808001-809000       * ---        II 809001-810000
##    [74360]        II 809001-810000       * ---        II 809001-810000
##             strand2 |   bin_id1   bin_id2     count  balanced probability
##               <Rle> | <numeric> <numeric> <numeric> <numeric>   <numeric>
##        [1]       * |       231       232         1       NaN 7.83580e-09
##        [2]       * |       231       236         2       NaN 2.81318e-08
##        [3]       * |       231       237         1       NaN 2.02960e-08
##        [4]       * |       231       239         2       NaN 6.73108e-08
##        [5]       * |       231       240         3       NaN 7.37336e-08
##        ...     ... .       ...       ...       ...       ...         ...
##    [74356]       * |      1038      1040         8 0.0472023 3.85638e-07
##    [74357]       * |      1038      1041         1       NaN 5.03006e-08
```

```
##     [74358]       * |      1039      1039         1       NaN 8.74604e-08
##     [74359]       * |      1039      1040         7       NaN 1.02111e-07
##     [74360]       * |      1040      1040         2 0.0411355 1.19216e-07
##              predicted      pvalue      qvalue logFoldChange
##              <numeric>   <numeric>   <numeric>     <numeric>
##         [1] 0.00369352 3.68670e-03 0.063385760       8.08079
##         [2] 0.01326033 8.71446e-05 0.001926954       7.23674
##         [3] 0.00956681 9.52120e-03 0.150288341       6.70775
##         [4] 0.03172791 4.92808e-04 0.009806734       5.97810
##         [5] 0.03475538 6.81713e-06 0.000173165       6.43158
##         ...        ...         ...         ...           ...
##     [74356]  0.1817758 2.51560e-11 1.07966e-09       5.45977
##     [74357]  0.0237099 2.34310e-02 3.38098e-01       5.39837
##     [74358]  0.0412257 4.03875e-02 5.49519e-01       4.60031
##     [74359]  0.0481315 1.13834e-13 5.77259e-12       7.18423
##     [74360]  0.0561941 1.52097e-03 2.79707e-02       5.15344
##     -------
##     regions: 802 ranges and 4 metadata columns
##     seqinfo: 16 sequences from an unspecified genome
```

References

10 Workflow 1: Distance-dependent interactions across yeast mutants

Aims

This chapter illustrates how to:

- Compute P(s) of several samples and compare them
- Compute distance-adjusted correlation between Hi-C datasets with `HiCRep`
- Perform differential interaction analysis between Hi-C datasets with `multiHiCcompare`

Datasets

We leverage seven yeast datasets in this notebook. They are all available from SRA:

- `SRR8769554`: WT yeast strain, G1 phase (rep1)
- `SRR10687276`: WT yeast strain, G1 phase (rep12)
- `SRR8769549`: WT yeast strain, G2/M phase (rep1)
- `SRR10687281`: WT yeast strain, G2/M phase (rep12)
- `SRR8769551`: *wpl1* mutant yeast strain, G2/M phase (rep1)
- `SRR10687278`: *wpl1* mutant yeast strain, G2/M phase (rep2)
- `SRR8769555`: *wpl1/eco1* mutant yeast strain, G2/M phase

10.1 Recovering data from SRA

The easiest for this is to directly fetch files from SRA from their FTP server. We can do so using the base `download.file` function.

> **Note**
>
> The next two code chunks illustrate how to do download and process Hi-C reads from SRA, but they are not actually executed when rendering this website as it would take a significant amount of time.

```
# !! This code is not actually executed !!
dir.create('data')
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/004/SRR8769554/SRR8769554_1.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/004/SRR8769554/SRR8769554_2.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/076/SRR10687276/SRR10687276_1.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/076/SRR10687276/SRR10687276_2.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/009/SRR8769549/SRR8769549_1.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/009/SRR8769549/SRR8769549_2.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/081/SRR10687281/SRR10687281_1.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/081/SRR10687281/SRR10687281_2.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/001/SRR8769551/SRR8769551_1.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/001/SRR8769551/SRR8769551_2.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/078/SRR10687278/SRR10687278_1.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR106/078/SRR10687278/SRR10687278_2.fastq
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/005/SRR8769555/SRR8769555_1.fastq.g
download.file("ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR876/005/SRR8769555/SRR8769555_2.fastq.g
```

10.2 Processing reads with HiCool

We will map each pair of fastqs on the yeast genome reference (`R64-1-1`) using `HiCool`.

```
# !! This code is not actually executed !!
library(HiCool)
samples <- c(
```

```
    'WT_G1_rep1',
    'WT_G1_rep2',
    'WT_G2M_rep1',
    'WT_G2M_rep2',
    'wpl1_G2M_rep1',
    'wpl1_G2M_rep2',
    'wpl1eco1_G2M'
)
purrr::map(samples, ~ HiCool(
    r1 = paste0('data/', .x, '_R1.fastq.gz'),
    r2 = paste0('data/', .x, '_R2.fastq.gz'),
    genome = 'R64-1-1',
    restriction = 'DpnII',
    iterative = FALSE,
    threads = 15,
    output = 'data/HiCool/',
    scratch = '/data/scratch/'
))
```

Processed samples are put in `data/HiCool` directory. `CoolFile` objects are pointers to individual contact matrices. We can create such objects by using the `importHiCoolFolder` utility function.

```
cfs <- list(
    WT_G1_rep1 = importHiCoolFolder('data/HiCool', 'GK8ISZ'),
    WT_G1_rep2 = importHiCoolFolder('data/HiCool', 'SWZTOO'),
    WT_G2M_rep1 = importHiCoolFolder('data/HiCool', '3KHHUE'),
    WT_G2M_rep2 = importHiCoolFolder('data/HiCool', 'UVNG7M'),
    wpl1_G2M_rep1 = importHiCoolFolder('data/HiCool', 'Q4KX6Z'),
    wpl1_G2M_rep2 = importHiCoolFolder('data/HiCool', '3NOL25'),
    wpl1eco1_G2M = importHiCoolFolder('data/HiCool', 'LHMXWE')
)
cfs
```

Now that these pointers have been defined, Hi-C contact matrices can be seamlessly imported in R with `import`.

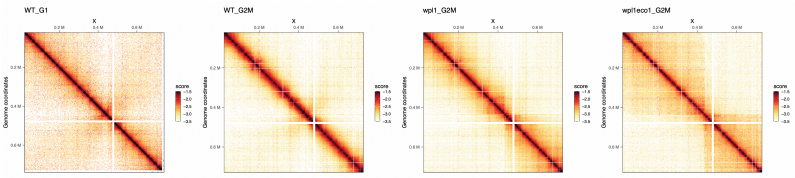
```
library(purrr)
library(HiCExperiment)
```

```
hics <- map(cfs, import)
hics
## $WT_G1_rep1
## `HiCExperiment` object with 5,454,145 contacts over 12,079 regions
## -------
## fileName: "../OHCA-data/HiCool/matrices/W303_G1_WT_rep1^mapped-S288c^GK8ISZ.mcool"
## focus: "whole genome"
## resolutions(5): 1000 2000 4000 8000 16000
## active resolution: 1000
## interactions: 3347524
## scores(2): count balanced
## topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
## pairsFile: ../OHCA-data/HiCool/pairs/W303_G1_WT_rep1^mapped-S288c^GK8ISZ.pairs
## metadata(3): log args stats
##
## $WT_G1_rep2
## `HiCExperiment` object with 12,068,214 contacts over 12,079 regions
## -------
## fileName: "../OHCA-data/HiCool/matrices/W303_G1_WT_rep2^mapped-S288c^SWZTO0.mcool"
## focus: "whole genome"
## resolutions(5): 1000 2000 4000 8000 16000
## active resolution: 1000
## interactions: 6756099
## scores(2): count balanced
## topologicalFeatures: compartments(0) borders(0) loops(0) viewpoints(0)
## pairsFile: ../OHCA-data/HiCool/pairs/W303_G1_WT_rep2^mapped-S288c^SWZTO0.pairs
## metadata(3): log args stats
##
## ...
```

10.3 Plotting chromosome-wide matrices of merged replicates

We can merge replicates with the `merge` function, and `map` the `plotMatrix` function over the resulting list of `HiCExperiments`.

```
library(HiContacts)
chr <- 'X'
merged_replicates <- list(
    WT_G1 = merge(hics[[1]][chr], hics[[2]][chr]),
    WT_G2M = merge(hics[[3]][chr], hics[[4]][chr]),
    wpl1_G2M = merge(hics[[5]][chr], hics[[6]][chr]),
    wpl1eco1_G2M = hics[[7]][chr]
)
library(dplyr)
library(ggplot2)
maps <- imap(merged_replicates, ~ plotMatrix(
    .x, use.scores = 'balanced', limits = c(-3.5, -1.5), caption = FALSE
) + ggtitle(.y))
cowplot::plot_grid(plotlist = maps, nrow = 1)
```



We can already note that long-range contacts seem to increase in frequency, in G2/M vs G1, in *wpl1* vs WT and in *wpl1/eco1* vs *wpl1*.

10.4 Compute P(s) per replicate and plot it

Still using the `map` function, we can compute average P(s) for each replicate.
Computation of the P(s) will take some time, as millions of pairs have to be imported in memory, but it will be accurate at the base resolution, rather than bin resolution from matrices.

> **Note**
>
> Since matrices were imported after `HiCool` processing with the `importHiCoolFolder`, the associated `.pairs` file has been automatically added to each `HiCExperiment` object!

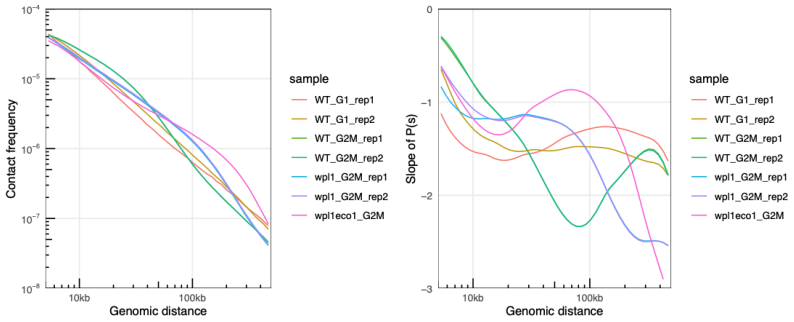
The computed P(s) is stored for each sample as a `tibble`.

```
pairsFile(hics[[1]])
ps <- imap(hics, ~ distanceLaw(.x) |> mutate(sample = .y))
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G1_WT_rep1^mapped-S288c^GK8ISZ.pairs
## |======================================================================| 100% 318 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G1_WT_rep2^mapped-S288c^SWZTO0.pairs
## |======================================================================| 100% 674 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G2M_WT_rep1^mapped-S288c^3KHHUE.pairs
## |======================================================================| 100% 709 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G2M_WT_rep2^mapped-S288c^UVNG7M.pairs
## |======================================================================| 100% 1683 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G2M_wpl1_rep1^mapped-S288c^Q4KX6Z.pai
## |======================================================================| 100% 1269 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G2M_wpl1_rep2^mapped-S288c^3N0L25.pai
## |======================================================================| 100% 1529 MB
## Importing pairs file ../OHCA-data/HiCool/pairs/W303_G2M_wpl1-eco1^mapped-S288c^LHMXWE.pai
## |======================================================================| 100% 1036 MB
ps[[1]]
## # A tibble: 133 x 6
##   binned_distance          p     norm_p norm_p_unity slope sample
##             <dbl>      <dbl>      <dbl>        <dbl> <dbl> <chr>
## 1               1 0.000154   0.000154         249.    0     WT_G1_rep1
## 2               2 0.0000563  0.0000563         91.2  0.702 WT_G1_rep1
## 3               3 0.0000417  0.0000417         67.5  0.699 WT_G1_rep1
## 4               4 0.00000835 0.00000835        13.5  0.696 WT_G1_rep1
## 5               5 0.00000501 0.00000501         8.10 0.693 WT_G1_rep1
## 6               6 0.00000250 0.00000250         4.05 0.690 WT_G1_rep1
## # ... with 127 more rows
```

We can bind all `tibbles` together and plot P(s) and their slope for each sample.

```
df <- list_rbind(ps)
plotPs(
    df, aes(x = binned_distance, y = norm_p,
    group = sample, color = sample)
)
plotPsSlope(
    df, aes(x = binned_distance, y = slope,
```

```
    group = sample, color = sample)
)
```



10.5 Correlation between replicates with `hicrep`

`hicrep` is a popular package to compute stratum-adjusted correlations between Hi-C datasets. "Stratum" refers to the distance from the main diagonal: with increase distance from the main diagonal, interactions of the DNA polymer are bound to decrease. `hicrep` computes a "per-stratum" correlation score and computes a weighted average correlation for entire chromosomes.

We can check the documentation for `hicrep` main function, `get.scc`. This tells us that `mat1` and `mat2` `n*n` intrachromosomal contact maps of raw `counts` should be provided. Fortunately, `HiCExperiment` objects can easily be coerced into actual dense matrices using `as.matrix()` function.

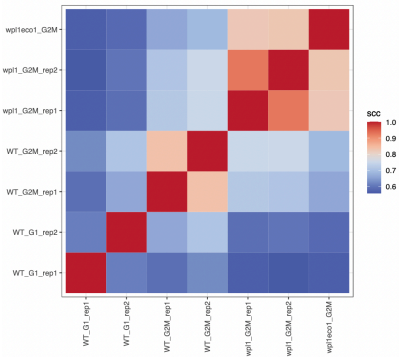
> **!** Important
>
> Make sure to use the `count` scores, which are required by `hicrep`.

We can calculate the overall stratum-corrected correlation score over the chromosome `IV` between the two `G2M` WT replicates.

```
library(hicrep)
scc <- get.scc(
    hics[['WT_G2M_rep1']]["IV"] |> as.matrix(use.scores = 'count'),
    hics[['WT_G2M_rep2']]["IV"] |> as.matrix(use.scores = 'count'),
    resol = 1000, h = 2, lbr = 5000, ubr = 50000
)
names(scc)
## [1] "corr" "wei"  "scc"  "std"
scc$scc
##           [,1]
## [1,] 0.9785691
```

This can be generalized to all pairwise combinations of Hi-C datasets.

```
library(purrr)
library(dplyr)
library(ggplot2)
mats <- map(hics, ~ .x["IV"] |> as.matrix(use.scores = 'count'))
df <- map(1:7, function(i) {
    map(1:7, function(j) {
        data.frame(
            i = names(hics)[i],
            j = names(hics)[j],
            scc = hicrep::get.scc(mats[[i]], mats[[j]], resol = 1000, h = 2, lbr = 5000, ubr
        ) |>
            mutate(i = factor(i, names(cfs))) |>
            mutate(j = factor(j, names(cfs)))
    }) |> list_rbind()
}) |> list_rbind()
ggplot(df, aes(x = i, y = j, fill = scc)) +
    geom_tile() +
    scale_x_discrete(guide = guide_axis(angle = 90)) +
    theme_bw() +
    coord_fixed(ratio = 1) +
    scale_fill_gradientn(colours = bgrColors())
```



We can even iterate over an extra level, to compute stratum-corrected correlation for all chromosomes. Here, we will only compute correlation scores between any sample and `WT_G2M_rep1` sample.

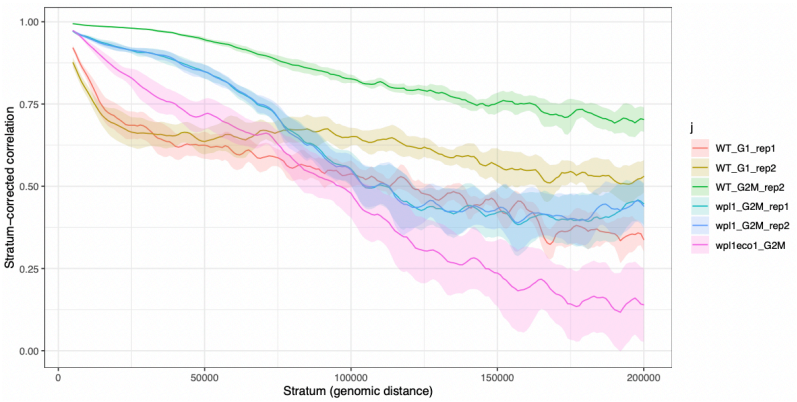
> **Parallelizing over chromosomes**
>
> `BiocParallel::bplapply()` replaces `purrr::map()` here, as it allows parallelization of independent correlation computation runs over multiple CPUs.

```
# Some chromosomes will be ignored as they are too small for this analysis
chrs <- c('II', 'IV', 'V', 'VII', 'VIII', 'IX', 'X', 'XI', 'XIII', 'XIV', 'XVI')
bpparam <- BiocParallel::MulticoreParam(workers = 6, progressbar = TRUE)
df <- BiocParallel::bplapply(chrs, function(CHR) {
    mats <- map(hics, ~ .x[CHR] |> interactions() |> gi2cm('count') |> cm2matrix())

        map(c(1, 2, 4, 5, 6, 7), function(j) {
            data.frame(
                chr = CHR,
                i = "WT_G2M_rep1",
                j = names(mats)[j],
                dist = seq(5000, 200000, 1000),
                scc = hicrep::get.scc(mats[["WT_G2M_rep1"]], mats[[j]], resol = 1000, h = 2,
            ) |> mutate(j = factor(j, names(mats)))
        }) |> list_rbind()

}, BPPARAM = bpparam) |> list_rbind()
```

A tiny bit of data wrangling will allow us to plot the mean

+/- confidence interval (90%) of stratum-adjusted correlations across the different chromosomes.

```
results <- group_by(df, j, dist) |>
    summarize(
        mean = Rmisc::CI(scc.corr, ci = 0.90)[2],
        CI_up = Rmisc::CI(scc.corr, ci = 0.90)[1],
        CI_down = Rmisc::CI(scc.corr, ci = 0.90)[3]
    )
ggplot(results, aes(x = dist, y = mean, ymax = CI_up, ymin = CI_down)) +
    geom_line(aes(col = j)) +
    geom_ribbon(aes(fill = j), alpha = 0.2, col = NA) +
    theme_bw() +
    labs(x = "Stratum (genomic distance)", y = 'Stratum-corrected correlation')
```



10.6 Differential interaction (DI) analysis with `multiHiCcompare`

We will now focus on the chromosome `XI` and identify differentially interacting (DI) loci between WT and *wpl1* mutant in G2/M.

To do this, we can use the `multiHiCcompare` package. The required input for the main `make_hicexp()` function is a `list` of raw counts for different samples/replicates, stored in data frames with four columns (`chr`, `start1`, `start2`, `count`). Although this data structure does not correspond to a standard

HiC format, it is easy to manipulate a `HiCExperiment` object to coerce it into such structure.

```
library(multiHiCcompare)
hics_list <- map(hics, ~ .x['XI'] |>
    zoom(2000) |>
    as.data.frame() |>
    select(start1, start2, count) |>
    mutate(chr = 1) |>
    relocate(chr)
)
mhicc <- make_hicexp(
    data_list = hics_list[c(3, 4, 5, 6)],
    groups = factor(c(1, 1, 2, 2)
), A.min = 1)
```

The `mhicc` object contains data over the chromosome `XI` binned at 2kb for two pairs of replicates (WT or `wpl1` G2/M HiC, each in duplicates):

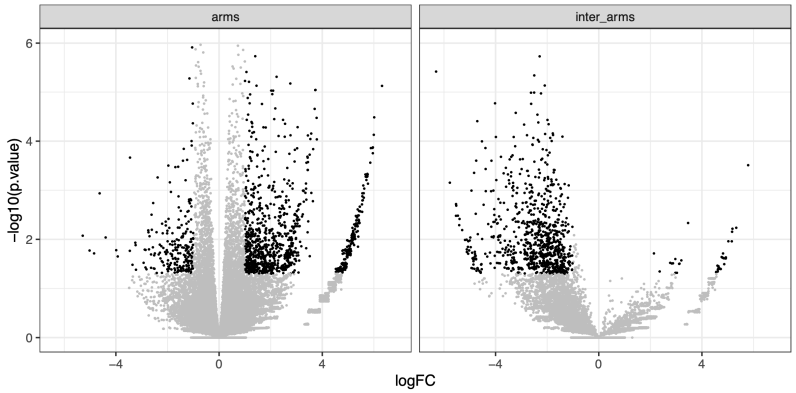
- Group1 contains WT data
- Group2 contains `wpl1` data

To identify differential interactions, the actual statistical comparison is performed with the `hic_exactTest()` function.

```
results <- cyclic_loess(mhicc, span = 0.2) |> hic_exactTest()
## |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=00s
## |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05s
results
## Hi-C Experiment Object
## 2 experimental groups
## Group 1 has 2 samples
## Group 2 has 2 samples
## Data has been normalized
```

The `results()` output is not very informative as it is. It requires a little bit of reformating to be able to extract valuable insights from it.

```
df <- left_join(results@hic_table, results(results)) |>
    mutate(dist = region2 - region1) |>
    mutate(group = case_when(
        region1 < 430000 & region2 > 450000 ~ 'inter_arms',
        region1 >= 430000 & region2 <= 450000 ~ 'at_centro',
        TRUE ~ 'arms'
    )) |>
    filter(group %in% c('arms', 'inter_arms')) |>
    mutate(sign = p.value <= 0.05 & abs(logFC) >= 1)
df
## chr region1 region2 D   IF1   IF2   IF3   IF4   logFC  logCPM  p.value    p.adj dist grou
##   1       1       1 0  6.16  2.09  7.96  5.43  0.5401 4.81329 5.38e-01 7.94e-01    0  arm
##   1       1    2001 1 16.38 10.25 12.96 12.16 -0.2257 5.82484 7.00e-01 8.81e-01 2000  arm
##   1       1    4001 2 41.41 40.72 84.41 45.14  0.5064 7.69885 5.94e-02 2.16e-01 4000  arm
##   1       1    6001 3 22.26 30.51 73.83 48.48  1.2726 8.10243 6.48e-07 5.83e-05 6000  arm
##   1       1    8001 4 26.63 31.20 33.39 25.92  0.0998 7.55207 8.02e-01 9.34e-01 8000  arm
## ...
ggplot(df, aes(x = logFC, y = -log10(p.value), col = sign)) +
    geom_point(size = 0.2) +
    theme_bw() +
    facet_wrap(~group) +
    ylim(c(0, 6)) +
    theme(legend.position = 'none') +
    scale_color_manual(values = c('grey', 'black'))
```

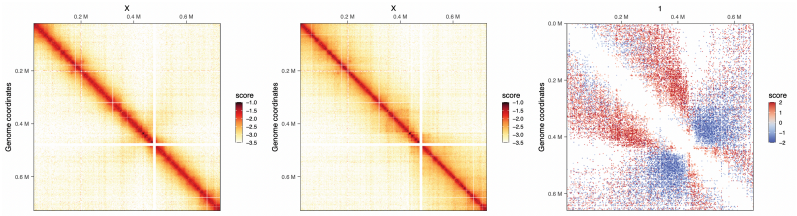


In this volcano plot, we can visually appreciate the fold-change of interaction frequency in WT or `wpl1`, for interactions constrained within the chromosome `XI` arms (left) or *spanning* the

chr. `XI` centromere (right). This clearly highlights that interactions within arms are increased in `wpl1` mutant while those spanning the centromere strongly decreased.

One of the strengths of `HiContacts` is that it can be leveraged to visualize any quantification related to genomic interactions as a HiC heatmap, since `plotMatrix` can take a `GInteractions` object with any score saved in `mcols` as input.

```
gis <- rename(df, seqnames1 = chr, start1 = region1, start2 = region2) |>
    mutate(
        seqnames2 = seqnames1,
        end1 = start1 + 1999,
        end2 = start2 + 1999
    ) |>
    filter(abs(logFC) >= 1) |>
    df2gi()
cowplot::plot_grid(
    plotMatrix(merged_replicates[['WT_G2M']], use.scores = 'balanced', limits = c(-3.5, -1),
    plotMatrix(merged_replicates[['wpl1_G2M']], use.scores = 'balanced', limits = c(-3.5, -1
    plotMatrix(gis, use.scores = 'logFC', scale = 'linear', limits = c(-2, 2), cmap = bgrCol
    align = "hv", axis = 'tblr', nrow = 1
)
```



References

11 Workflow 2: Chromosome compartment cohesion upon mitosis entry

Aims

This chapter illustrates how to:

- Annotate compartments for a list of HiC experiments
- Generate saddle plots for a list of HiC experiments
- Quantify changes in interactions between compartments between different timepoints

Datasets

We leverage five chicken datasets in this notebook, published in @Gibcus_2018. They are all available from the 4DN data portal using the `fourDNData` package.

- `4DNES9LEZXN7`: chicken cell culture blocked in G2
- `4DNESNWWIFZU`: chicken cell culture released from G2 block (5min)
- `4DNESGDXKM2I`: chicken cell culture released from G2 block (10min)
- `4DNESIR416OW`: chicken cell culture released from G2 block (15min)
- `4DNESS8PTK6F`: chicken cell culture released from G2 block (30min)

11.1 Importing data

The 4DN consortium provides access to the datasets published in @Gibcus_2018. in `R`, they can be obtained thanks to the `fourDNData` gateway package.

> ⚠ Beware
>
> The first time the following chunk of code is executed, it will cache a large amount of data (mostly consisting of contact matrices stored in `.mcool` files).

```
library(HiCExperiment)
library(fourDNData)
library(BiocParallel)
samples <- list(
    '4DNES9LEZXN7' = 'G2 block',
    '4DNESNWWIFZU' = 'prophase (5m)',
    '4DNESGDXKM2I' = 'prophase (10m)',
    '4DNESIR416OW' = 'prometaphase (15m)',
    '4DNESS8PTK6F' = 'prometaphase (30m)'
)
bpparam <- MulticoreParam(workers = 5, progressbar = TRUE)
hics <- bplapply(names(samples), fourDNHiCExperiment, BPPARAM = bpparam)
## Fetching local Hi-C contact map from Bioc cache
## Fetching local compartments bigwig file from Bioc cache
## Fetching local insulation bigwig file from Bioc cache
## Fetching local borders bed file from Bioc cache
## Importing contacts in memory
## |=================================================| 100%
##
## ...
```

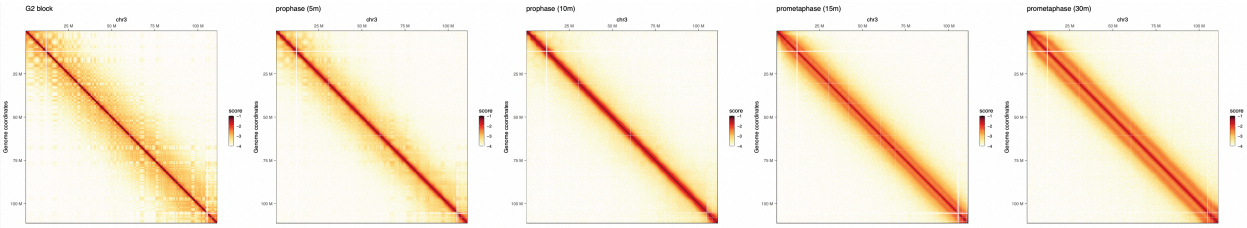
```
names(hics) <- samples
hics[["G2 block"]]
## `HiCExperiment` object with 150,494,008 contacts over 4,109 regions
## -------
## fileName: "/home/rsg/.cache/R/fourDNData/25c33c9d826f_4DNFIT479GDR.mcool"
## focus: "whole genome"
```

```
## resolutions(13): 1000 2000 ... 5000000 10000000
## active resolution: 250000
## interactions: 7262748
## scores(2): count balanced
## topologicalFeatures: compartments(891) borders(3465)
## pairsFile: N/A
## metadata(3): 4DN_info eigens diamond_insulation
```

11.2 Plotting whole chromosome matrices

We can visualize the five different Hi-C maps on the entire chromosome `3` with `HiContacts` by iterating over each of the `HiCExperiment` objects.

```
library(purrr)
library(HiContacts)
pl <- imap(hics, ~ .x['chr3'] |>
    zoom(100000) |>
    plotMatrix(use.scores = 'balanced', limits = c(-4, -1), caption = FALSE) +
    ggtitle(.y)
)
library(cowplot)
plot_grid(plotlist = pl, nrow = 1)
```



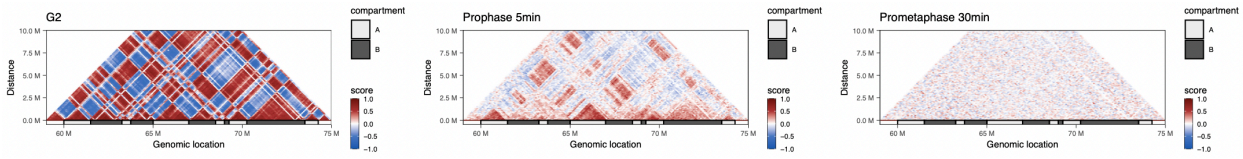
This highlights the progressive remodeling of chromatin into condensed chromosomes, starting as soon as 5' after release from G2 phase.

11.3 Zooming on a chromosome section

Zooming on a chromosome section, we can plot the Hi-C autocorrelation matrix for each timepoint. These matrices are generally used to highlight the overall correlation of interaction profiles between different segments of a chromosome section (see Chapter 5 for more details).

```
## --- Format compartment positions of chr. 4 segment
.chr <- 'chr4'
.start <- 59000000L
.stop <- 75000000L
library(GenomicRanges)
coords <- GRanges(paste0(.chr, ':', .start, '-', .stop))
compts_df <- topologicalFeatures(hics[["G2 block"]], "compartments") |>
    subsetByOverlaps(coords, type = 'within') |>
    as.data.frame()
compts_gg <- geom_rect(
    data = compts_df,
    mapping = aes(xmin = start, xmax = end, ymin = -500000, ymax = 0, alpha = compartment),
    col = 'black', inherit.aes = FALSE
)

## --- Subset contact matrices to chr. 4 segment and computing autocorrelation scores
g2 <- hics[["G2 block"]] |>
    subsetByOverlaps(coords) |>
    zoom(100000) |>
    autocorrelate()
pro5 <- hics[["prophase (5m)"]] |>
    subsetByOverlaps(coords) |>
    zoom(100000) |>
    autocorrelate()
pro30 <- hics[["prometaphase (30m)"]] |>
    subsetByOverlaps(coords) |>
    zoom(100000) |>
    autocorrelate()

## --- Plot autocorrelation matrices
plot_grid(
    plotMatrix(
```

```
        g2,
        use.scores = 'autocorrelated',
        scale = 'linear',
        limits = c(-1, 1),
        cmap = bwrColors(),
        maxDistance = 10000000,
        caption = FALSE
    ) + ggtitle('G2') + compts_gg,
    plotMatrix(
        pro5,
        use.scores = 'autocorrelated',
        scale = 'linear',
        limits = c(-1, 1),
        cmap = bwrColors(),
        maxDistance = 10000000,
        caption = FALSE
    ) + ggtitle('Prophase 5min') + compts_gg,
    plotMatrix(
        pro30,
        use.scores = 'autocorrelated',
        scale = 'linear',
        limits = c(-1, 1),
        cmap = bwrColors(),
        maxDistance = 10000000,
        caption = FALSE
    ) + ggtitle('Prometaphase 30min') + compts_gg,
    nrow = 1
)
```



These correlation matrices suggest that there are two different regimes of chromatin compartment remodeling in this chromosome section:

1. Correlation scores between genomic bins within the compartment A remain positive 5' after G2 release (albeit

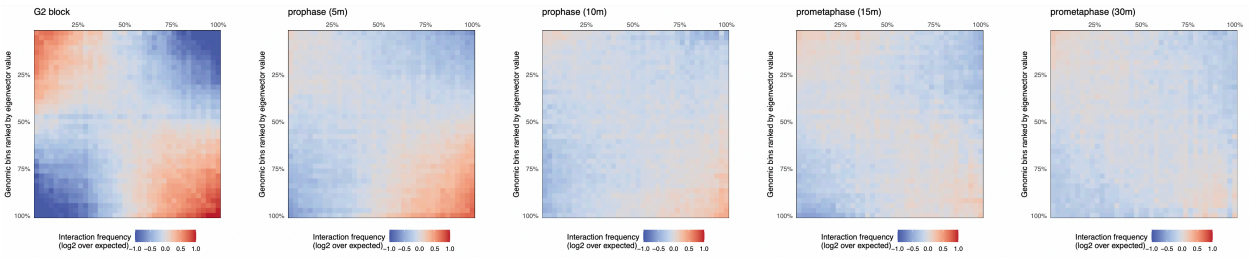
reduced compared to G2 block) and eventually become null 30' after G2 release.
2. Correlation scores between genomic bins within the compartment B are overall null as soon as 5' after G2 release.

11.4 Generating saddle plots

Saddle plots are typically used to measure the `observed` vs. `expected` interaction scores within or between genomic loci belonging to A and B compartments. Here, they can be used to check whether the two regimes of chromatin compartment remodeling are observed genome-wide.

Non-overlapping genomic windows are grouped by `nbins` quantiles (typically between 10 and 50 bins) according to their A/B compartment eigenvector value, from lowest eigenvector values (i.e. strongest B compartments) to highest eigenvector values (i.e. strongest A compartments). The average `observed` vs. `expected` interaction scores are computed for pairwise eigenvector quantiles and plotted in a 2D heatmap.

```
pl <- imap(hics, ~ plotSaddle(.x, nbins = 38, BPPARAM = bpparam) + ggtitle(.y))
plot_grid(plotlist = pl, nrow = 1)
```



These plots confirm the previous observation made on chr. 4 and reveal that intra-B compartment interactions are generally lost 5' after G2 release, while intra-A interactions take up to 15' after G2 release to disappear.

> **Beware**
>
> The `plotSaddle()` function requires an eigenvector corresponding to A/B compartments. In this example, this eigenvector is recovered from the 4DN data portal. If not already available, this eigenvector can be computed from the contact matrix using the `getCompartments()` function.

11.5 Quantifying interactions within and between compartments

We can leverage the replicate-merged contact matrices to quantify the interaction frequencies within A or B compartments or between A and B comaprtments, at different timepoints.

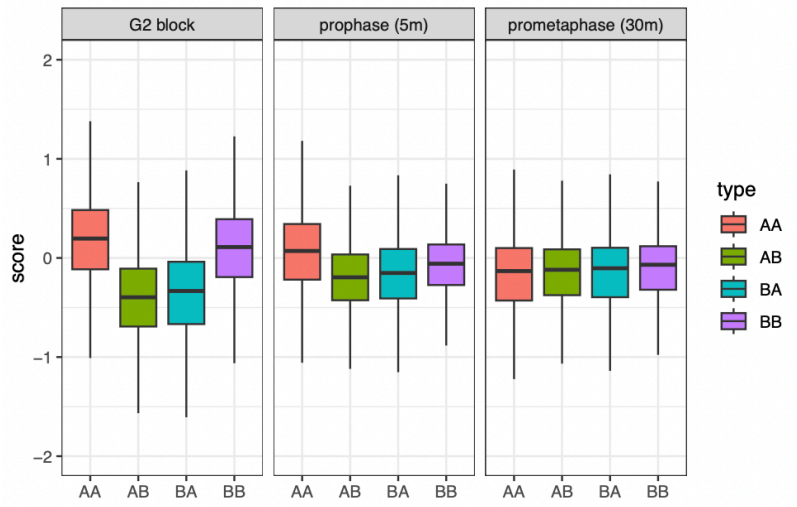
We can use the A/B compartment annotations obtained at the `G2 block` timepoint and extract `O/E` (observed vs expected) scores for interactions within A or B compartments or between A and B compartments, at different timepoints.

```
## --- Extract the A/B compartments identified in G2 block
compts <- topologicalFeatures(hics[["G2 block"]], "compartments")
compts$ID <- paste0(compts$compartment, seq_along(compts))

## --- Iterate over timepoints to extract `detrended` (O/E) scores and
##     compartment annotations
library(plyranges)
df <- imap(hics[c(1, 2, 5)], ~ {
    ints <- cis(.x) |> ## Filter out trans interactions
        detrend() |> ## Compute O/E scores
        interactions() ## Recover interactions
    ints$comp_first <- join_overlap_left(anchors(ints, "first"), compts)$ID
    ints$comp_second <- join_overlap_left(anchors(ints, "second"), compts)$ID
    tibble(
        sample = .y,
        bin1 = ints$comp_first,
        bin2 = ints$comp_second,
```

```
        dist = pairdist(ints),
        OE = ints$detrended
    ) |>
        filter(dist > 5e6) |>
        mutate(type = case_when(
            grepl('A', bin1) & grepl('A', bin2) ~ 'AA',
            grepl('B', bin1) & grepl('B', bin2) ~ 'BB',
            grepl('A', bin1) & grepl('B', bin2) ~ 'AB',
            grepl('B', bin1) & grepl('A', bin2) ~ 'BA'
        )) |>
        filter(bin1 != bin2)
}) |> list_rbind() |> mutate(
    sample = factor(sample, names(hics)[c(1, 2, 5)])
)
```

We can now plot the changes in O/E scores for intra-A, intra-B, A-B or B-A interactions, splitting boxplots by timepoint.

```
ggplot(df, aes(x = type, y = OE, group = type, fill = type)) +
    geom_boxplot(outlier.shape = NA) +
    facet_grid(~sample) +
    theme_bw() +
    ylim(c(-2, 2))
```



This visualization suggests that interactions between genomic loci belonging to the B compartment are lost more rapidly than

those between genomic loci belonging to the A compartment, when cells are released from G2 to enter mitosis.

References

12 Workflow 3: Inter-centromere interactions in yeast

Aims

This chapter illustrates how to plot the aggregate signal over pairs of genomic ranges, in this case pairs of yeast centromeres.

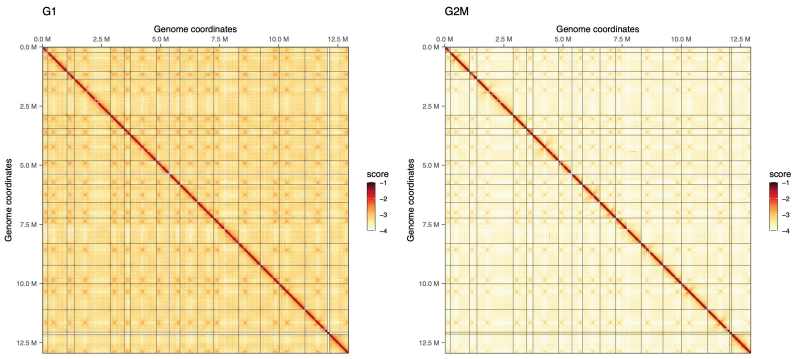
Datasets

We leverage two yeast datasets in this notebook.

- One from a WT yeast strain in G1 phase
- One from a WT yeast strain in G2/M phase

12.1 Importing Hi-C data and plotting contact matrices

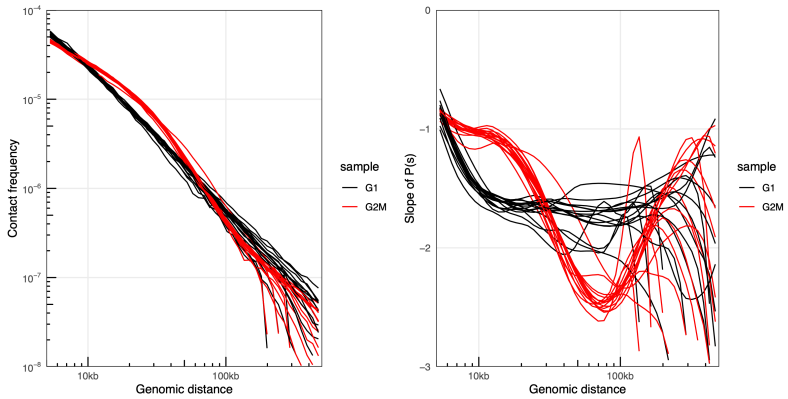
```
library(HiContacts)
library(purrr)
library(ggplot2)
hics <- list(
    'G1' = import('/home/rsg/repos/OHCA-data/S288c_G1.mcool', resolution = 4000),
    'G2M' = import('/home/rsg/repos/OHCA-data/S288c_G2M.mcool', resolution = 4000)
)
imap(hics, ~ plotMatrix(
    .x, use.scores = 'balanced', limits = c(-4, -1), caption = FALSE
) + ggtitle(.y))
```



We can visually appreciate that inter-chromosomal interactions, notably between centromeres, are less prominent in G2/M.

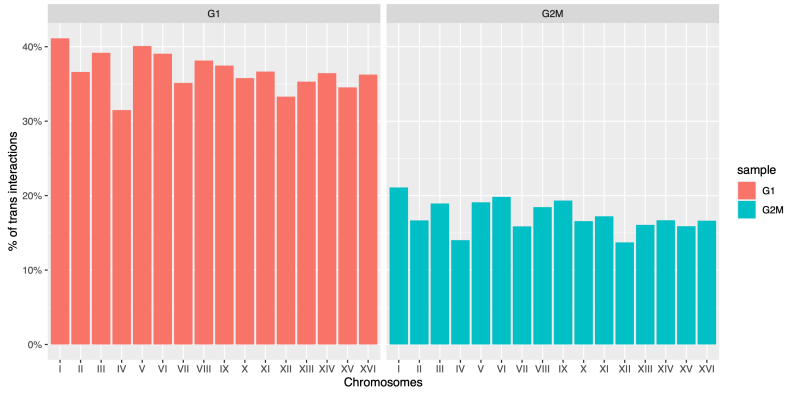
12.2 Checking P(s) and cis/trans interactions ratio

```
library(dplyr)
pairs <- list(
    'G1' = PairsFile('/home/rsg/repos/OHCA-data/S288c_G1.pairs'),
    'G2M' = PairsFile('/home/rsg/repos/OHCA-data/S288c_G2M.pairs')
)
ps <- imap_dfr(pairs, ~ distanceLaw(.x, by_chr = TRUE) |>
    mutate(sample = .y)
)
plotPs(ps, aes(x = binned_distance, y = norm_p, group = interaction(sample, chr), color = sa
    scale_color_manual(values = c('black', 'red'))
plotPsSlope(ps, ggplot2::aes(x = binned_distance, y = slope, group = interaction(sample, chr
    scale_color_manual(values = c('black', 'red'))
```



This confirms that interactions in cells synchronized in G2/M are enriched for 10-30kb-long interactions.

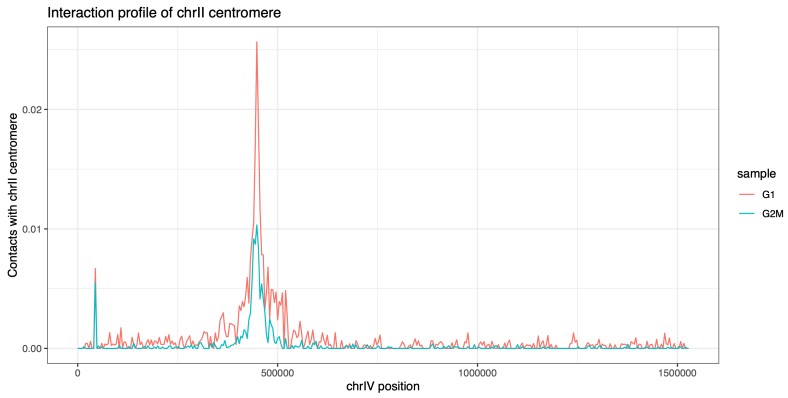
```
ratios <- imap_dfr(hics, ~ cisTransRatio(.x) |> mutate(sample = .y))
ggplot(ratios, aes(x = chr, y = trans_pct, fill = sample)) +
    geom_col() +
    labs(x = 'Chromosomes', y = "% of trans interactions") +
    scale_y_continuous(labels = scales::percent) +
    facet_grid(~sample)
```



We can also highlight that trans (inter-chromosomal) interactions are proportionally decreasing in G2/M-synchronized cells.

12.3 Centromere virtual 4C profiles

```
data(centros_yeast)
v4c_centro <- imap_dfr(hics, ~ virtual4C(.x, resize(centros_yeast[2], 8000)) |>
    as_tibble() |>
    mutate(sample = .y) |>
    filter(seqnames == 'IV')
)
ggplot(v4c_centro, aes(x = start, y = score, colour = sample)) +
    geom_line() +
    theme_bw() +
    labs(
        x = "chrIV position",
        y = "Contacts with chrII centromere",
        title = "Interaction profile of chrII centromere"
    )
```



12.4 Aggregated 2D signal over all pairs of centromeres

We can start by computing all possible pairs of centromeres.

```
centros_pairs <- lapply(1:length(centros_yeast), function(i) {
    lapply(1:length(centros_yeast), function(j) {
        S4Vectors::Pairs(centros_yeast[i], centros_yeast[j])
```

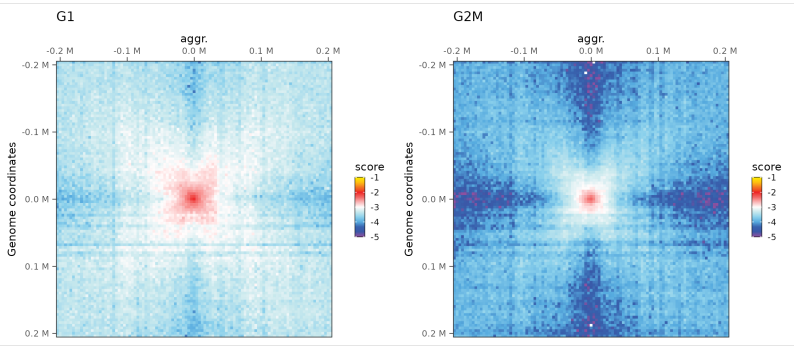
```
    })
}) |>
    do.call(c, args = _) |>
    do.call(c, args = _) |>
    InteractionSet::makeGInteractionsFromGRangesPairs()
centros_pairs <- centros_pairs[anchors(centros_pairs, 'first') != anchors(centros_pairs, 'se

centros_pairs
## GInteractions object with 240 interactions and 0 metadata columns:
##         seqnames1       ranges1     seqnames2       ranges2
##             <Rle>     <IRanges>         <Rle>     <IRanges>
##     [1]         I 151583-151641 ---        II 238361-238419
##     [2]         I 151583-151641 ---       III 114322-114380
##     [3]         I 151583-151641 ---        IV 449879-449937
##     [4]         I 151583-151641 ---         V 152522-152580
##     [5]         I 151583-151641 ---        VI 147981-148039
##     ...       ...           ... ...       ...           ...
##   [236]       XVI 556255-556313 ---        XI 440229-440287
##   [237]       XVI 556255-556313 ---       XII 151366-151424
##   [238]       XVI 556255-556313 ---      XIII 268222-268280
##   [239]       XVI 556255-556313 ---       XIV 628588-628646
##   [240]       XVI 556255-556313 ---        XV 326897-326955
##   -------
##   regions: 16 ranges and 0 metadata columns
##   seqinfo: 17 sequences (1 circular) from R64-1-1 genome
```

Then we can aggregate the Hi-C signal over each pair of centromeres.

```
aggr_maps <- purrr::imap(hics, ~ {
    aggr <- aggregate(.x, centros_pairs, maxDistance = 1e999)
    plotMatrix(
        aggr, use.scores = 'balanced', limits = c(-5, -1),
        cmap = HiContacts::rainbowColors(),
        caption = FALSE
    ) + ggtitle(.y)
})
## Going through preflight checklist...
## Parsing the entire contact matrice as a sparse matrix...
```

```
## Modeling distance decay...
## Filtering for contacts within provided targets...
## Going through preflight checklist...
## Parsing the entire contact matrice as a sparse matrix...
## Modeling distance decay...
## Filtering for contacts within provided targets...
```

```
cowplot::plot_grid(plotlist = aggr_maps, nrow = 1)
```

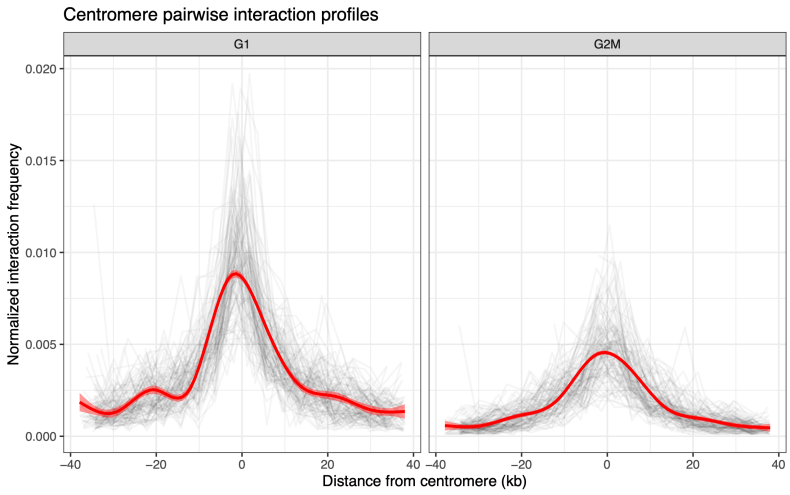


12.5 Aggregated 1D interaction profile of centromeres

One can generalize the previous virtual 4C plot, by extracting the interaction profile between all possible pairs of centromeres in each dataset.

```
df <- map_dfr(1:{length(centros_yeast)-1}, function(i) {
    centro1 <- resize(centros_yeast[i], fix = 'center', 8000)
    map_dfr({i+1}:length(centros_yeast), function(j) {
        centro2 <- resize(centros_yeast[j], fix = 'center', 80000)
        gi <- GInteractions(centro1, centro2)
        imap_dfr(hics, ~ .x[gi] |>
            interactions() |>
```

```
            as_tibble() |>
            mutate(
                sample = .y,
                center = center2 - start(resize(centro2, fix = 'center', 1))
            ) |>
            select(sample, seqnames1, seqnames2, center, balanced)
        )
    })
})
p <- ggplot(df, aes(x = center/1e3, y = balanced)) +
    geom_line(aes(group = interaction(seqnames1, seqnames2)), alpha = 0.03, col = "black") +
    geom_smooth(col = "red", fill = "red") +
    theme_bw() +
    theme(legend.position = 'none') +
    labs(
        x = "Distance from centromere (kb)", y = "Normalized interaction frequency",
        title = "Centromere pairwise interaction profiles"
    ) +
    facet_grid(~sample)
```



References