**Этот отчет предоставляет детальное описание процесса анализа данных Hi-C, выполненного в рамках обучения по анализу данных Hi-C. В процессе работы использовались различные инструменты и пакеты для подготовки, обработки и визуализации данных. Основной целью было воспроизведение анализа, описанного в руководствах и документации, а также представление результатов в виде контактных матриц и их дальнейшего анализа.**

**Был установлен Docker на локальной машине для обеспечения изолированного окружения. Это позволило избежать проблем с зависимостями и обеспечить воспроизводимость анализа.**

**Использовался Docker-образ duplexa/4dn-hic:v42, включающий все необходимые инструменты и зависимости для работы с Hi-C данными.**

# Hi-C Data Analysis Bootcamp

## Pointers for Offline Walk-through

During the bootcamp, users were given access to linux servers where

* docker was installed,
* conda was installed,
* a conda enivronment was set up with a number of dependencies installed, including juypter notebook,
* higlass-manager was installed,
* and sample data was downloaded.

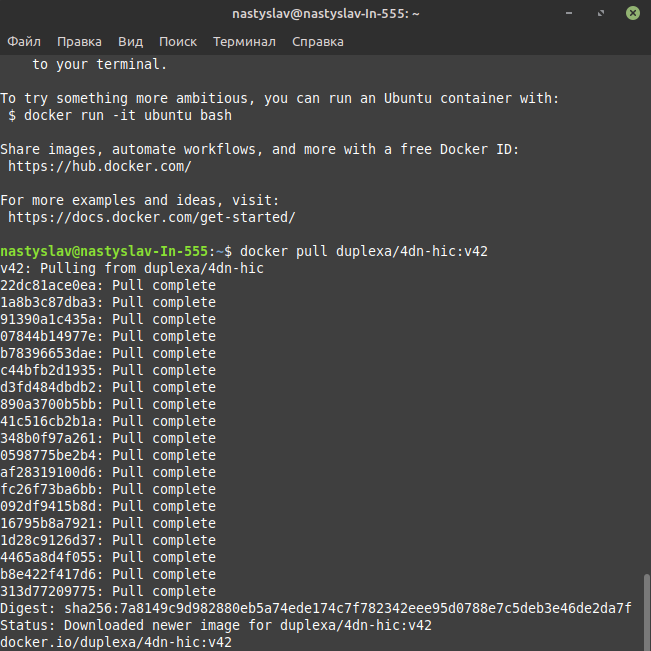
You can set up a similar environment and walk through the hands-on sessions of the bootcamp by following the instructions below. Allow 30G of storage for all files used in the tutorial.

**From fastqs to contact matrices**

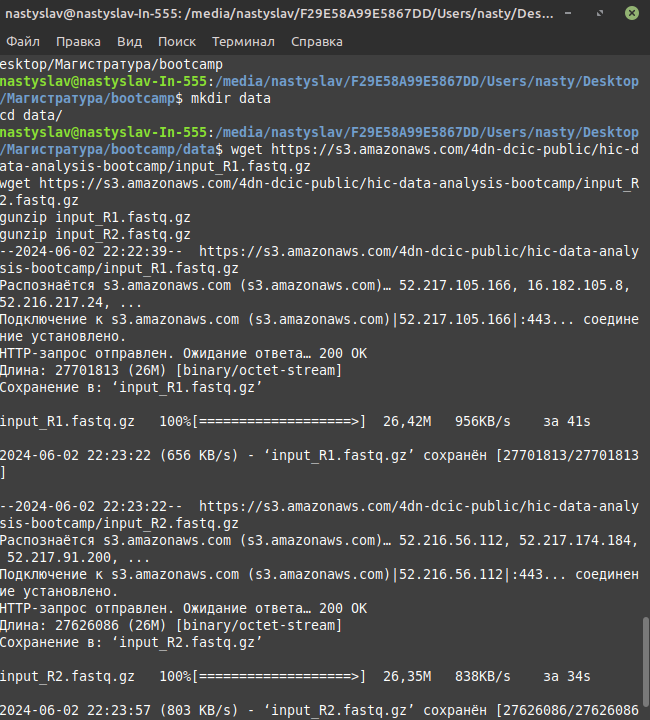
1. Install docker, if you have not already done so. (Docker is a lighter alternative to virtual machines.)
2. Pull the docker image: docker pull duplexa/4dn-hic:v42. This docker image contains a number of software that have been pre-installed for HiC data processing.

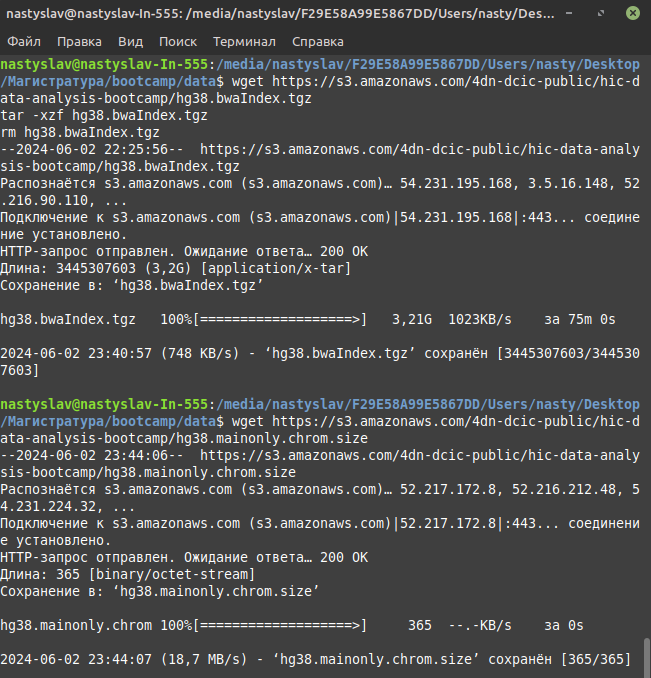
**Conda была установлена для управления пакетами и создания виртуального окружения.**

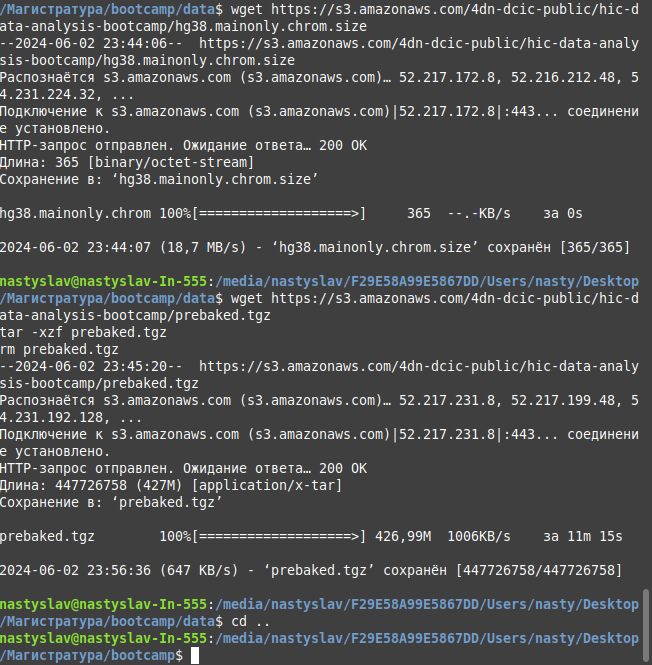
**Создано новое окружение и установлены необходимые пакеты, включая juypter notebook и higlass-manager.**

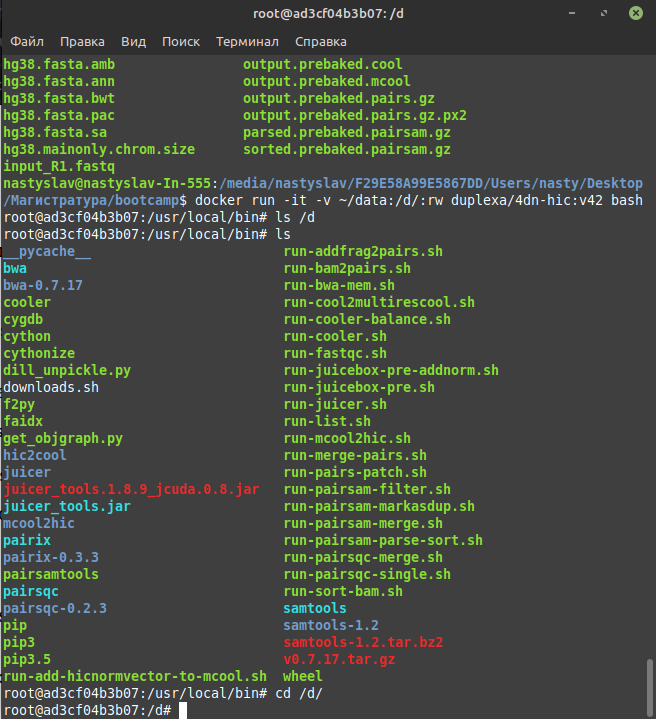
1. Download the sample data for this session under your home directory to "~/data/" (or edit the commands on the slides accordingly, if you prefer a different directory).

**Данные были загружены в домашний каталог ~/data/ для дальнейшей обработки.**

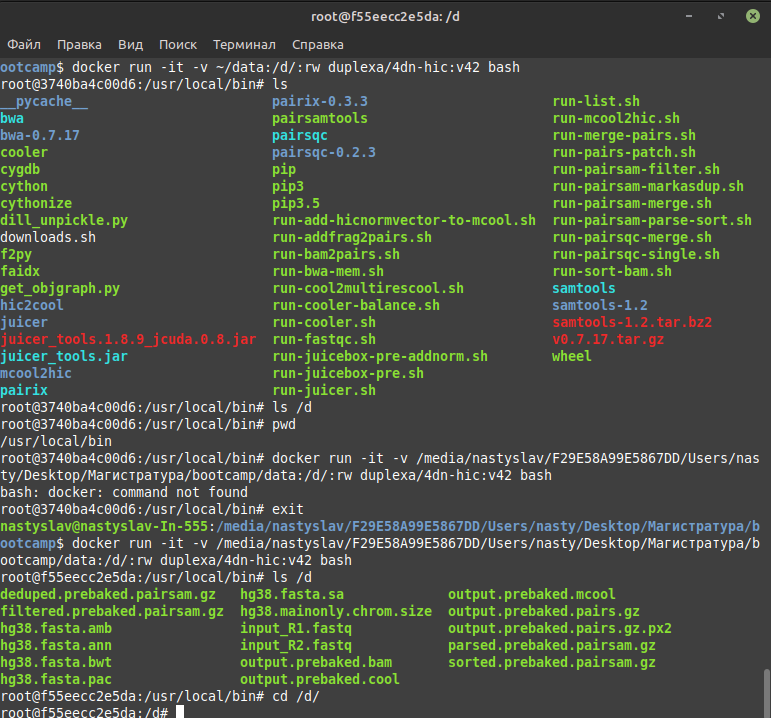


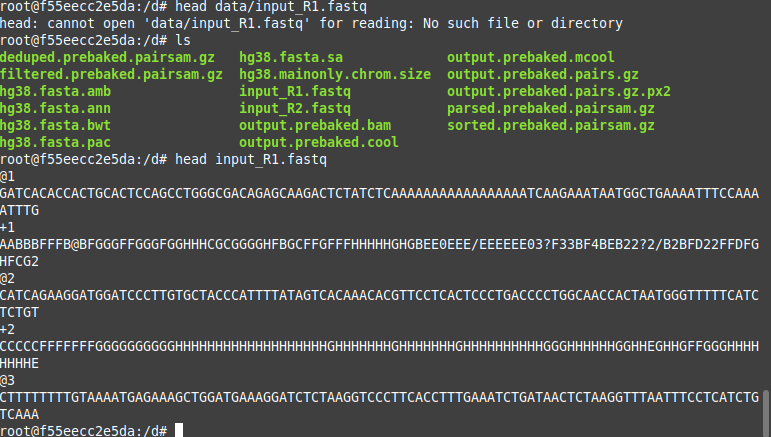


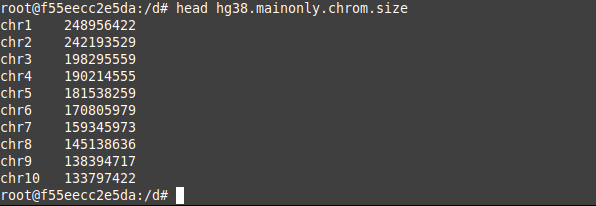


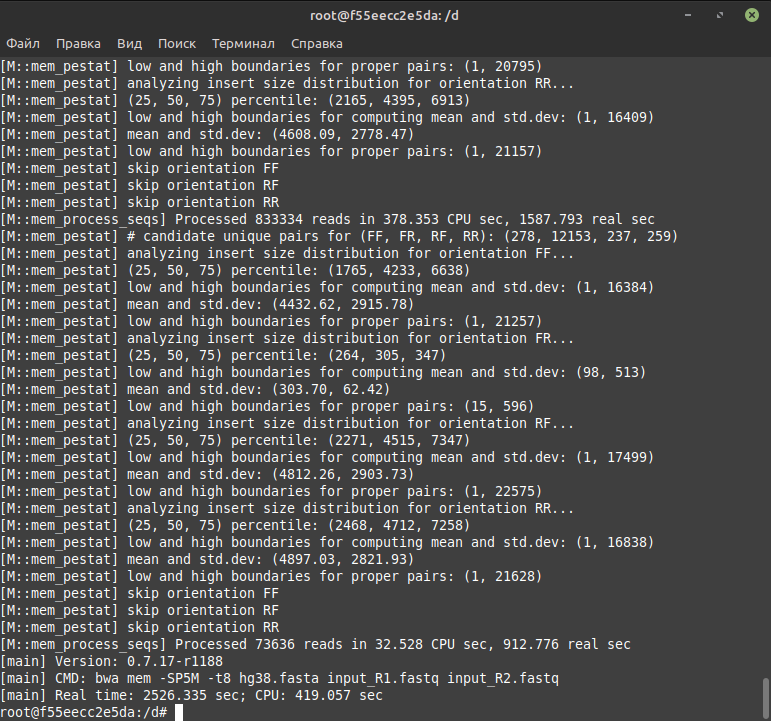


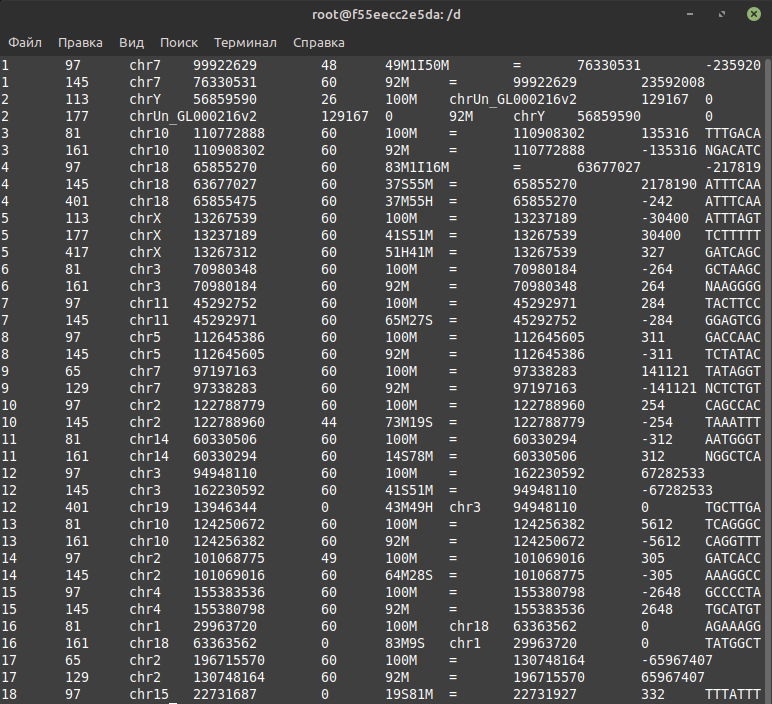
**путь был неверный, исправила**

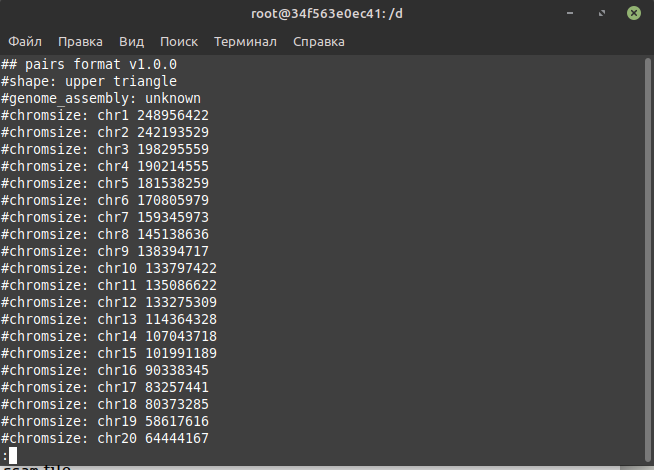


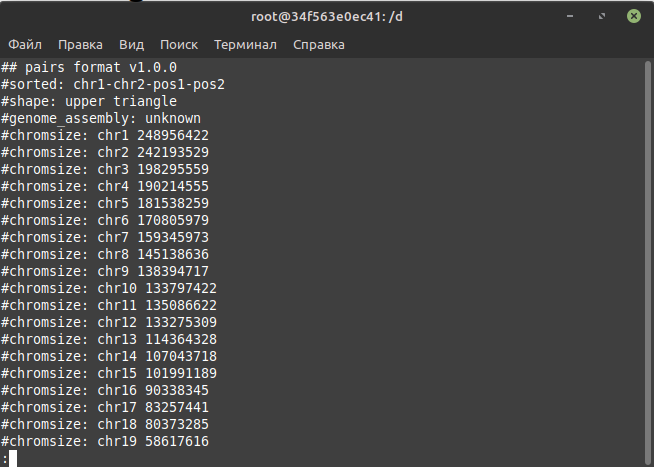


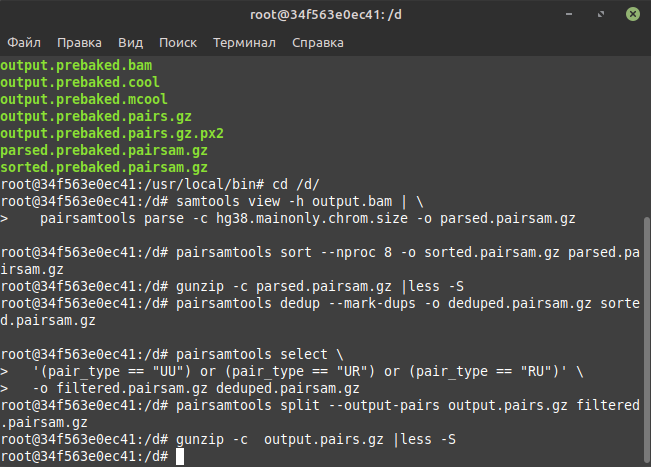


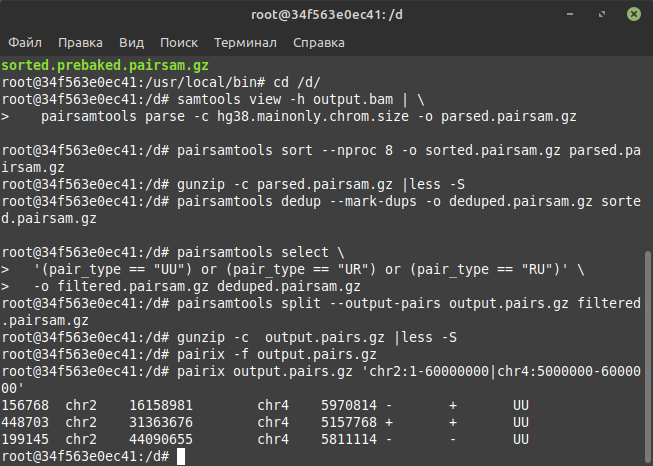


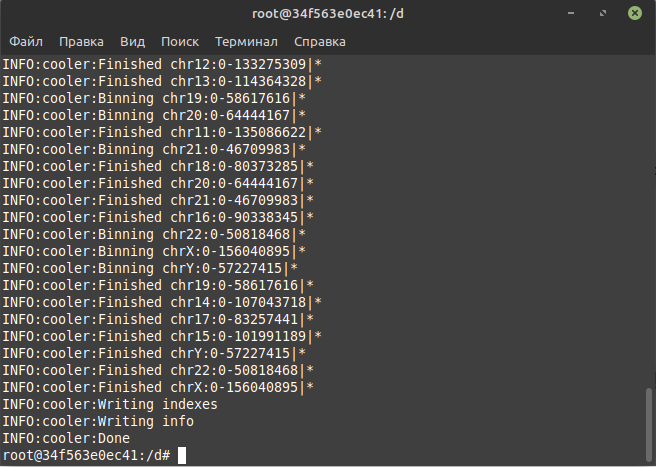


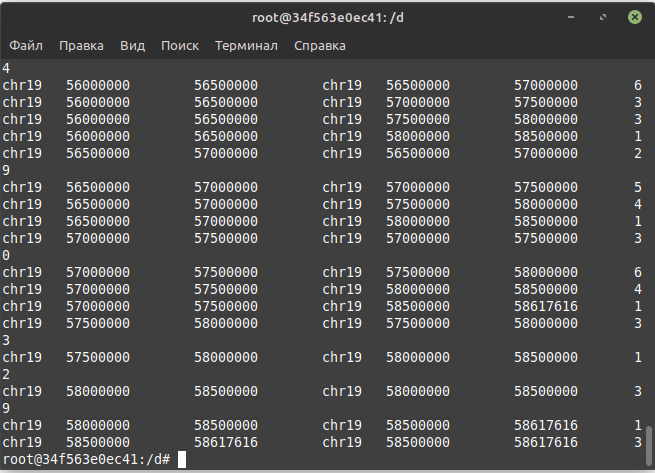


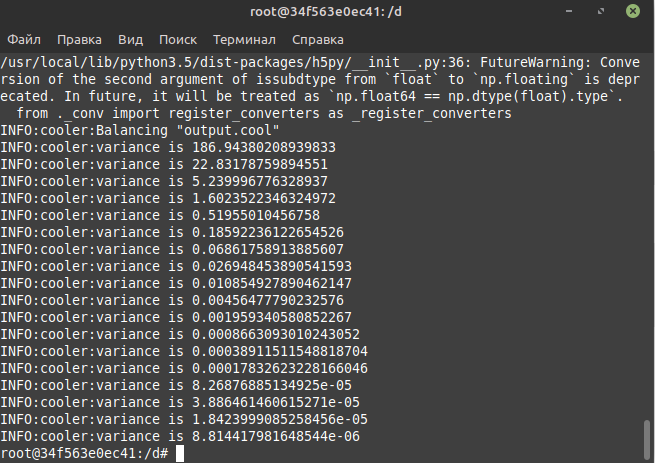


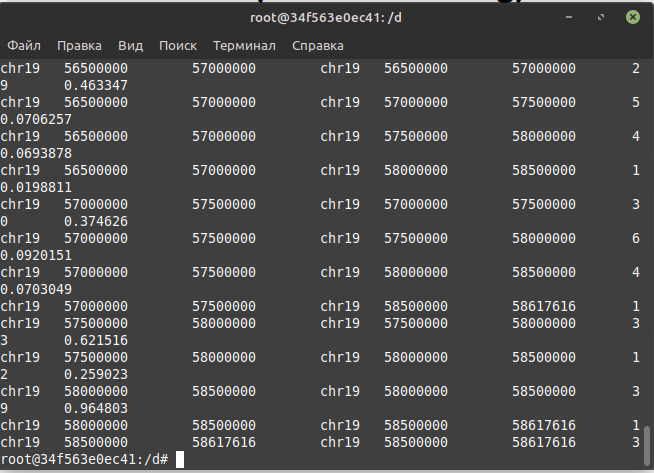


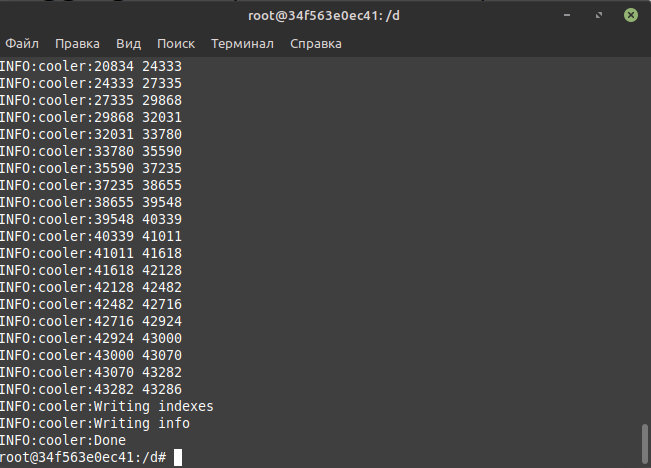










Go back to the "notebooks" directory and activate the environment to run the jupyter notebook

# Построение контактной матрицы:

# Использован инструмент cooler для агрегации списка пар ридов в файл формата cool.

# Геном был разбит на бины фиксированного размера 1 Мб с помощью команды cooler makebins.

# Построена контактная матрица для данных GM12878 на разрешении 1 Мб с помощью команды cooler cload.

# Cooler command line interface

If you type cooler at the command line with no arguments or with -h or --help you'll get the following quick reference of available subcommands.

%%bash

​

cooler -h

Usage: cooler [OPTIONS] COMMAND [ARGS]...

Type -h or --help after any subcommand for more information.

Options:

-v, --verbose Verbose logging.

-d, --debug On error, drop into the post-mortem debugger shell.

-V, --version Show the version and exit.

-h, --help Show this message and exit.

Commands:

balance Out-of-core matrix balancing.

cload Create a cooler from genomic pairs and bins.

coarsen Coarsen a cooler to a lower resolution.

csort Sort and index a contact list.

digest Generate fragment-delimited genomic bins.

dump Dump a cooler's data to a text stream.

ls List all coolers inside a file.

cp Copy a cooler from one file to another or within the same file.

ln Create a hard link to a cooler (rather than a true copy) in the...

mv Rename a cooler within the same file.

tree Display a file's data hierarchy.

attrs Display a file's attribute hierarchy.

info Display a cooler's info and metadata.

load Create a cooler from a pre-binned matrix.

makebins Generate fixed-width genomic bins.

merge Merge multiple coolers with identical axes.

show Display and browse a cooler in matplotlib.

zoomify Generate a multi-resolution cooler file by coarsening.

For more information about a specific subcommand, type cooler <subcommand> -h to display the help text.

%%bash

​

cooler info -h

Usage: cooler info [OPTIONS] COOL\_PATH

Display a cooler's info and metadata.

COOL\_PATH : Path to a COOL file or cooler URI.

Options:

-f, --field TEXT Print the value of a specific info field.

-m, --metadata Print the user metadata in JSON format.

-o, --out TEXT Output file (defaults to stdout)

-h, --help Show this message and exit.

## Example

Let's try it.

%%bash

​

cooler info data/Rao2014-GM12878-MboI-allreps-filtered.1000kb.cool

{

"bin-size": 1000000,

"bin-type": "fixed",

"creation-date": "2016-02-25T21:05:29.075865",

"format-url": "https://github.com/mirnylab/cooler",

"format-version": 2,

"genome-assembly": "hg19",

"id": null,

"library-version": "0.3.0",

"nbins": 3114,

"nchroms": 25,

"nnz": 4150156

}

%%bash

​

cooler info -f bin-size data/Rao2014-GM12878-MboI-allreps-filtered.1000kb.cool

1000000

%%bash

​

cooler info -m data/Rao2014-GM12878-MboI-allreps-filtered.1000kb.cool

{

"publication": "",

"QC": {

"double-sided": {

"total": 3390352656,

"valid": 3147639590,

"filtered-invalid": {

"removed-self-circles": 1741768,

"removed-error-pair": 6074295,

"removed-dangling-ends": 234897003

},

"filtered-valid": {

"removed-duplicate": 110650005,

"removed-start-near-rsite": "",

"removed-outlier-fragment": 151337031,

"removed-large-small-pair": 657466

}

},

"pre-filtering": {

"total": 5332721651,

"double-sided": 3390352656,

"unused": 0,

"single-sided": 1942368995

},

"post-filtering": {

"total": 2884995088,

"cis": 2085711027,

"trans": 799284061

}

},

"enzyme": "MboI",

"cell-type": "GM12878",

"species": "Homo sapiens",

"sex": "F"

}

## Aggregate a list of read pairs into a cool file

To make a contact matrix, we need

1. A list of read pairs representing captured contacts.
2. A segmentation of the genome into bins by which we aggregate (bin) the read pair counts.

For(1), we will start with a very small subsample of 100,000 read pairs from GSM1551552 (Rao et al, GM12878). The fields of the file are readID, strand1, chrom1, pos1, frag1, strand2, chrom2, pos2, frag2, mapq1, mapq2.

%%bash

​

zcat data/GSM1551552\_HIC003\_merged\_nodups.txt.subset.gz | head

D260LACXX130602:2:2315:7361:72358 0 1 85378 186 16 1 591085 1097 0 0

D258GACXX130605:8:2316:2958:10584 0 1 728403 1418 16 1 719104 1406 20 10

C24LCACXX130513:8:2315:5697:82732 0 1 758309 1523 0 1 43498676 121266 68 120

D260LACXX130602:2:2202:16754:100485 16 1 890311 1801 16 1 993887 2056 165 18

D260LACXX130602:2:2309:8547:48542 0 1 925938 1866 16 1 1034493 2117 178 60

C24LCACXX130513:8:2312:14225:27548 0 1 941657 1904 0 1 1620964 3551 178 0

D258GACXX130605:8:2308:5276:50416 0 1 992230 2049 0 1 1255504 2556 175 178

C24LCACXX130513:8:1308:7340:36704 0 1 1070613 2191 16 1 1070941 2193 153 40

C24LCACXX130513:8:2104:5074:54778 16 1 1137145 2329 16 1 201625081 455535 144 178

C24LCACXX130513:8:1302:14464:56521 0 1 1170252 2381 16 1 1353433 2788 178 22

This data was mapped to the Broad's b37 assembly and uses ENSEMBL-style chromosome names (1..22, X, Y, MT) instead of the UCSC-style (chr1..chr22, chrX, chrY, chrM).

We provide a chromosome sizes file with the chromosomes we want to use in a desired order. Make sure the name style of the chromsizes file matches the name style of the pairs file! The following is the b37 chromosome sizes file with chromosomes in a "natural" semantic order, leaving out the unlocalized and unplaced scaffolds.

%%bash

​

cat data/b37.chrom.sizes.reduced

1 249250621

2 243199373

3 198022430

4 191154276

5 180915260

6 171115067

7 159138663

8 146364022

9 141213431

10 135534747

11 135006516

12 133851895

13 115169878

14 107349540

15 102531392

16 90354753

17 81195210

18 78077248

19 59128983

20 63025520

21 48129895

22 51304566

X 155270560

Y 59373566

MT 16569

We also need to decide how we want to bin the contacts. Usually, we choose a fixed bin size or "resolution". Another option for Hi-C data is to use restriction fragment-delimited genomic bins based on the restriction enzyme used in the experiment. cooler allows for any binning scheme you like, as long as you provide it as a bin table. We can store a bin table in a simple BED file using the makebins command.

%%bash

​

cooler makebins -h

Usage: cooler makebins [OPTIONS] CHROMSIZES\_PATH BINSIZE

Generate fixed-width genomic bins. Output a genome segmentation at a fixed

resolution as a BED file.

CHROMSIZES\_PATH : UCSC-like chromsizes file, with chromosomes in desired

order.

BINSIZE : Resolution (bin size) in base pairs <int>.

Options:

-o, --out TEXT Output file (defaults to stdout)

-h, --help Show this message and exit.

If you have the FASTA sequence of the reference genome, you can also "digest" it to create a bin table of fragments.

%%bash

​

cooler digest -h

Usage: cooler digest [OPTIONS] CHROMSIZES\_PATH FASTA\_PATH ENZYME

Generate fragment-delimited genomic bins. Output a genome segmentation of

restriction fragments as a BED file.

CHROMSIZES\_PATH : UCSC-like chromsizes file, with chromosomes in desired

order.

FASTA\_PATH : Genome assembly FASTA file or folder containing FASTA files

(uncompressed).

ENZYME : Name of restriction enzyme

Options:

-o, --out TEXT Output file (defaults to stdout)

-h, --help Show this message and exit.

%%bash

​

CHROMSIZES\_FILE='data/b37.chrom.sizes.reduced'

​

cooler makebins --out bins.1000kb.bed $CHROMSIZES\_FILE 1000000

​

# what's in the file?

head bins.1000kb.bed

1 0 1000000

1 1000000 2000000

1 2000000 3000000

1 3000000 4000000

1 4000000 5000000

1 5000000 6000000

1 6000000 7000000

1 7000000 8000000

1 8000000 9000000

1 9000000 10000000

Note that there is a convenient syntax to specify a fixed-resolution bin table, so you rarely need to generate one manually:

<chromsizes\_path>:<binsize-in-bp>

e.g. The bin table above can be specified as data/b37.chrom.sizes.reduced:1000000.

### Index-free loading

New in v0.7.7

Pairs files no longer need to be indexed or sorted. Use the cooler cload pairs command.

%%bash

​

cooler cload pairs -h

Usage: cooler cload pairs [OPTIONS] BINS PAIRS\_PATH COOL\_PATH

Bin any text file or stream of pairs.

Pairs data need not be sorted. Accepts compressed files. To pipe input

from stdin, set PAIRS\_PATH to '-'.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS\_PATH : Path to contacts (i.e. read pairs) file.

COOL\_PATH : Output COOL file path or URI.

Options:

--metadata TEXT Path to JSON file containing user metadata.

--assembly TEXT Name of genome assembly (e.g. hg19, mm10)

-c1, --chrom1 INTEGER chrom1 field number (one-based) [required]

-p1, --pos1 INTEGER pos1 field number (one-based) [required]

-c2, --chrom2 INTEGER chrom2 field number (one-based) [required]

-p2, --pos2 INTEGER pos2 field number (one-based) [required]

--chunksize INTEGER Number of input lines to load at a time

-0, --zero-based Positions are zero-based [default: False]

--comment-char TEXT Comment character that indicates lines to

ignore. [default: #]

--tril-action [reflect|drop] How to handle lower triangle records.

'reflect': make lower triangle records upper

triangular. Use this if your input data comes

only from a unique half of a symmetric matrix

(but may not respect the specified chromosome

order). 'drop': discard all lower triangle

records. Use this if your input data has

mirror duplicates, i.e. is derived from a

complete symmetric matrix. [default: reflect]

-h, --help Show this message and exit.

%%bash

# Note that the input pairs file happens to be space-delimited, so we convert to tab-delimited with `tr`.

CHROMSIZES\_FILE='data/b37.chrom.sizes.reduced'

BINSIZE=1000000

PAIRS\_FILE='data/GSM1551552\_HIC003\_merged\_nodups.txt.subset.gz'

OUTPUT\_FILE='test.cool'

​

zcat $PAIRS\_FILE \

| tr ' ' '\t' \

| cooler cload pairs -c1 3 -p1 4 -c2 7 -p2 8 $CHROMSIZES\_FILE:$BINSIZE - $OUTPUT\_FILE

INFO:cooler:Writing chunk 0: /tmp/tmphi277xgr.multi.cool::0

INFO:cooler:Creating cooler at "/tmp/tmphi277xgr.multi.cool::/0"

INFO:cooler:Writing chroms

INFO:cooler:Writing bins

INFO:cooler:Writing pixels

INFO:cooler:Writing indexes

INFO:cooler:Writing info

INFO:cooler:Done

INFO:cooler:Merging into test.cool

INFO:cooler:Creating cooler at "test.cool::/"

INFO:cooler:Writing chroms

INFO:cooler:Writing bins

INFO:cooler:Writing pixels

INFO:cooler:nnzs: [46598]

INFO:cooler:current: [46598]

INFO:cooler:Writing indexes

INFO:cooler:Writing info

INFO:cooler:Done

There are benefits to sorting and indexing pairs. See below.

## Text export

The cooler dump command lets us print the data back out as text with several formatting and annotation options. It also accepts range queries, both intra- and inter-chromosomal.

%%bash

​

cooler dump -h

Usage: cooler dump [OPTIONS] COOL\_PATH

Dump a contact matrix. Print the contents of a COOL file to tab-delimited

text.

COOL\_PATH : Path to COOL file or Cooler URI.

Options:

-t, --table [chroms|bins|pixels]

Which table to dump. Choosing 'chroms' or

'bins' will cause all pixel-related options

to be ignored. Note that dumping 'pixels'

will only provide data for the upper

triangle of the contact matrix. [default:

pixels]

--header Print the header of column names as the

first row. [default: False]

-k, --chunksize INTEGER Sets the amount of pixel data loaded from

disk at one time. Can affect the performance

of joins on high resolution datasets.

Default is to load as many rows as there are

bins.

-r, --range TEXT The coordinates of a genomic region shown

along the row dimension, in UCSC notation.

(Example: chr1:10,000,000-11,000,000). If

omitted, the entire contact matrix is

printed.

-r2, --range2 TEXT The coordinates of a genomic region shown

along the column dimension. If omitted, the

column range is the same as the row range.

-b, --balanced / --no-balance Apply balancing weights to data. This will

print an extra column called `balanced`

[default: False]

--join Print the full chromosome bin coordinates

instead of bin IDs. This will replace the

`bin1\_id` column with `chrom1`, `start1`,

and `end1`, and the `bin2\_id` column with

`chrom2`, `start2` and `end2`. [default:

False]

--annotate TEXT Join additional columns from the bin table

against the pixels. Provide a comma

separated list of column names (no spaces).

The merged columns will be suffixed by '1'

and '2' accordingly.

-o, --out TEXT Output text file If .gz extension is

detected, file is written using zlib.

Default behavior is to stream to stdout.

-h, --help Show this message and exit.

%%bash

​

cooler dump -t chroms test.cool

1 249250621

2 243199373

3 198022430

4 191154276

5 180915260

6 171115067

7 159138663

8 146364022

9 141213431

10 135534747

11 135006516

12 133851895

13 115169878

14 107349540

15 102531392

16 90354753

17 81195210

18 78077248

19 59128983

20 63025520

21 48129895

22 51304566

X 155270560

Y 59373566

MT 16569

%%bash

​

cooler dump -t bins test.cool | head

1 0 1000000

1 1000000 2000000

1 2000000 3000000

1 3000000 4000000

1 4000000 5000000

1 5000000 6000000

1 6000000 7000000

1 7000000 8000000

1 8000000 9000000

1 9000000 10000000

%%bash

​

cooler dump -t pixels --header test.cool | head

bin1\_id bin2\_id count

0 0 3

0 1 4

0 43 1

0 155 1

0 229 1

0 437 1

0 492 1

0 493 1

0 666 1

%%bash

​

cooler dump -t pixels --header --join test.cool | head

chrom1 start1 end1 chrom2 start2 end2 count

1 0 1000000 1 0 1000000 3

1 0 1000000 1 1000000 2000000 4

1 0 1000000 1 43000000 44000000 1

1 0 1000000 1 155000000 156000000 1

1 0 1000000 1 229000000 230000000 1

1 0 1000000 2 187000000 188000000 1

1 0 1000000 2 242000000 243000000 1

1 0 1000000 2 243000000 243199373 1

1 0 1000000 3 172000000 173000000 1

%%bash

​

cooler dump -t pixels -r 10:10,000,000-20,000,000 -r2 10:30,000,000-80,000,000 --header --join test.cool | head

chrom1 start1 end1 chrom2 start2 end2 count

10 10000000 11000000 10 70000000 71000000 1

10 11000000 12000000 10 30000000 31000000 1

10 11000000 12000000 10 35000000 36000000 1

10 11000000 12000000 10 42000000 43000000 2

10 11000000 12000000 10 43000000 44000000 1

10 11000000 12000000 10 52000000 53000000 1

10 11000000 12000000 10 58000000 59000000 1

10 12000000 13000000 10 30000000 31000000 1

10 12000000 13000000 10 45000000 46000000 1

%%bash

​

cooler dump -t pixels --header --balanced test.cool | head

Balancing weights not found

Oops! Our contact matrix isn't balanced yet. Let's do that next.

## Balancing

Matrix balancing normalization, i.e. iterative correction.

We usually normalize or "correct" Hi-C using a technique called matrix balancing. This involves finding a set of weights or biases 𝑏𝑖

for each bin 𝑖

such that

𝑁𝑜𝑟𝑚𝑎𝑙𝑖𝑧𝑒𝑑[𝑖,𝑗]=𝑂𝑏𝑠𝑒𝑟𝑣𝑒𝑑[𝑖,𝑗]⋅𝑏[𝑖]⋅𝑏[𝑗],

such that the marginals (i.e., row/column sums) of the global contact matrix are flat and equal.

cooler balance will store the pre-computed balancing weights in the bin table as an extra column called weight.

Note that whole-genome matrix balancing on a high resolution matrix requires iterative computations on a matrix that may not fit in computer memory, even in sparse form. Our "out-of-core" method performs the calculations by splitting and loading the data into smaller chunks and combining the partial results afterwards.

%%bash

​

cooler balance -h

Usage: cooler balance [OPTIONS] COOL\_PATH

Out-of-core contact matrix balancing.

Assumes uniform binning. See the help for various filtering options to

ignore poorly mapped bins.

COOL\_PATH : Path to a COOL file.

Options:

-p, --nproc INTEGER Number of processes to split the work

between. [default: 8]

-c, --chunksize INTEGER Control the number of pixels handled by each

worker process at a time. [default:

10000000]

--mad-max INTEGER Ignore bins from the contact matrix using

the 'MAD-max' filter: bins whose log

marginal sum is less than ``mad-max`` median

absolute deviations below the median log

marginal sum of all the bins in the same

chromosome. [default: 5]

--min-nnz INTEGER Ignore bins from the contact matrix whose

marginal number of nonzeros is less than

this number. [default: 10]

--min-count INTEGER Ignore bins from the contact matrix whose

marginal count is less than this number.

[default: 0]

--blacklist PATH Path to a 3-column BED file containing

genomic regions to mask out during the

balancing procedure, e.g. sequence gaps or

regions of poor mappability.

--ignore-diags INTEGER Number of diagonals of the contact matrix to

ignore, including the main diagonal.

Examples: 0 ignores nothing, 1 ignores the

main diagonal, 2 ignores diagonals (-1, 0,

1), etc. [default: 2]

--ignore-dist INTEGER Distance in bp to ignore.

--tol FLOAT Threshold value of variance of the marginals

for the algorithm to converge. [default:

1e-05]

--max-iters INTEGER Maximum number of iterations to perform if

convergence is not achieved. [default: 200]

--cis-only Calculate weights against intra-chromosomal

data only instead of genome-wide.

--trans-only Calculate weights against inter-chromosomal

data only instead of genome-wide.

--name TEXT Name of column to write to. [default:

weight]

-f, --force Overwrite the target dataset, 'weight', if

it already exists.

--check Check whether a data column 'weight' already

exists.

--stdout Print weight column to stdout instead of

saving to file.

--convergence-policy [store\_final|store\_nan|discard|error]

What to do with weights when balancing

doesn't converge in max\_iters. [default:

store\_final]

-h, --help Show this message and exit.

cooler balance iterates until the balanced marginals (i.e. row sums of the balanced matrix) are sufficiently flat (the variance falls below the limit tol).

%%bash

​

cooler balance -p 10 -c 10000 test.cool

INFO:cooler:Balancing "test.cool"

INFO:cooler:variance is 108.70508940352958

INFO:cooler:variance is 11.544325707535195

INFO:cooler:variance is 2.6406181035180114

INFO:cooler:variance is 0.7820569377009834

INFO:cooler:variance is 0.2430786648548748

INFO:cooler:variance is 0.08214976966884285

INFO:cooler:variance is 0.028125223955516747

INFO:cooler:variance is 0.01007790709086642

INFO:cooler:variance is 0.003641581079865631

INFO:cooler:variance is 0.001351764078013928

INFO:cooler:variance is 0.0005055339361931292

INFO:cooler:variance is 0.00019213135598238304

INFO:cooler:variance is 7.348487864297036e-05

INFO:cooler:variance is 2.838381218829712e-05

INFO:cooler:variance is 1.1018544001827941e-05

INFO:cooler:variance is 4.304041056888988e-06

%%bash

​

cooler dump --header --balanced test.cool | head

bin1\_id bin2\_id count balanced

0 0 3 0.0619576

0 1 4 0.0917378

0 43 1 0.0246198

0 155 1 0.0223535

0 229 1 0.0292275

0 437 1 0.0364777

0 492 1 0.0238998

0 493 1 0.0843047

0 666 1 0.023308

## Display the contact matrix

You can also use the cooler show function to produce images of the contact matrix. Requires the matplotlib Python package.

%%bash

​

cooler show -h

Usage: cooler show [OPTIONS] COOL\_PATH RANGE

Display a contact matrix. Display a region of a contact matrix stored in a

COOL file.

Arguments:

COOL\_PATH : Path to a COOL file or Cooler URI.

RANGE : The coordinates of the genomic region to display, in UCSC

notation. Example: chr1:10,000,000-11,000,000

Options:

-r2, --range2 TEXT The coordinates of a genomic region shown

along the column dimension. If omitted, the

column range is the same as the row range.

Use to display asymmetric matrices or trans

interactions.

-b, --balanced Show the balanced contact matrix. If not

provided, display the unbalanced counts.

-o, --out TEXT Save the image of the contact matrix to a

file. If not specified, the matrix is

displayed in an interactive window. The

figure format is deduced from the extension

of the file, the supported formats are png,

jpg, svg, pdf, ps and eps.

--dpi INTEGER The DPI of the figure, if saving to a file

-s, --scale [linear|log2|log10]

Scale transformation of the colormap:

linear, log2 or log10. Default is log10.

-f, --force Force display very large matrices (>=10^8

pixels). Use at your own risk as it may

cause performance issues.

--zmin FLOAT The minimal value of the color scale. Units

must match those of the colormap scale. To

provide a negative value use a equal sign

and quotes, e.g. -zmin='-0.5'

--zmax FLOAT The maximal value of the color scale. Units

must match those of the colormap scale. To

provide a negative value use a equal sign

and quotes, e.g. -zmax='-0.5'

--cmap TEXT The colormap used to display the contact

matrix. See the full list at [http://matplotl](http://matplotl/)

ib.org/examples/color/colormaps\_reference.ht

ml

--field TEXT Pixel values to display. [default: count]

-h, --help Show this message and exit.

Here's the undersampled dataset.

%%bash

​

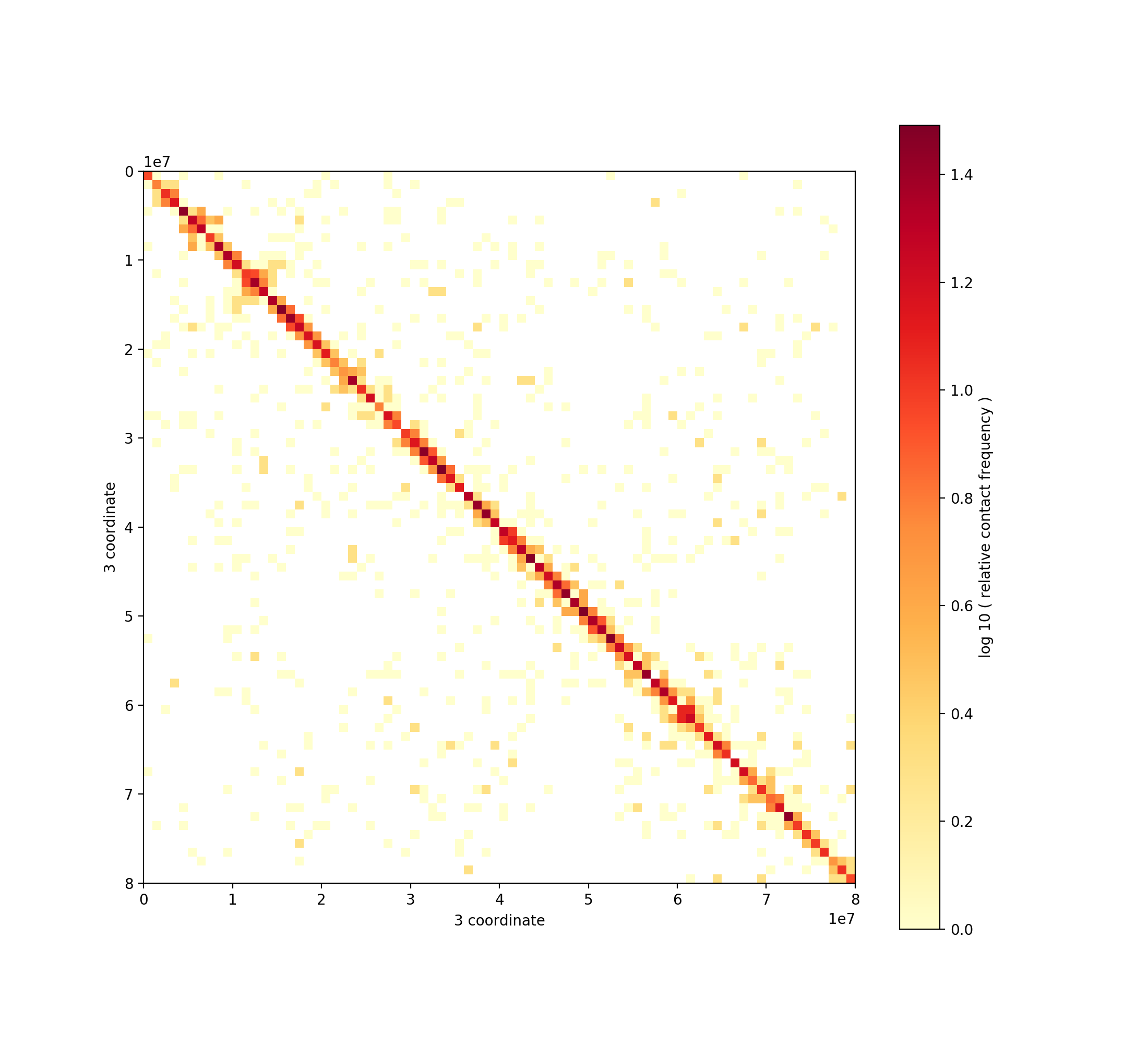
cooler show --out test.png --dpi 200 test.cool 3:0-80,000,000

WARNING:py.warnings:/net/proteome/home/nezar/local/devel/hicsparse/cooler/cli/show.py:31: RuntimeWarning: divide by zero encountered in log10

mat = np.log10(mat)

from IPython.display import Image

Image('test.png')



Here's what the full one looks like.

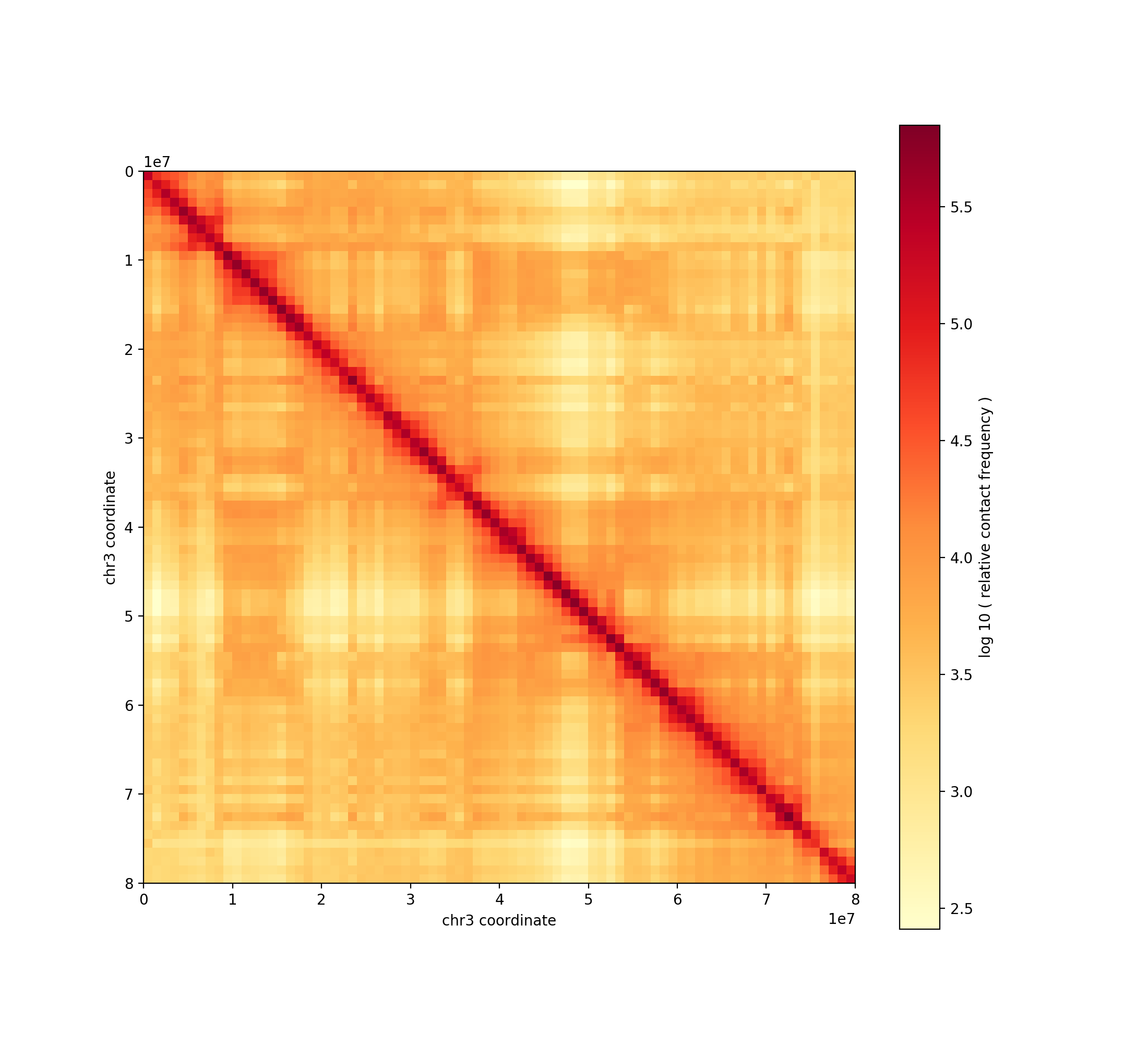
%%bash

​

cooler show --out test2.png --dpi 200 data/Rao2014-GM12878-MboI-allreps-filtered.1000kb.cool chr3:0-80,000,000

from IPython.display import Image

Image('test2.png')



## Load a contact list indexed with [Pairix](https://github.com/4dn-dcic/pairix)\*

Next, we need to make sure our pairs file is properly formatted and indexed. We use the cooler csort command to do this.

What does it do?

Given a chromosome order, it creates a new pairs file with the following properties:

1. Consistently ordered mates: mates of every interchromsomal pair are properly "flipped" in order to respect the requested order of the chromosomes. For intrachromosomal pairs, mates are flipped such that pos1 is always less than or equal to pos2. As a result, the data will have an upper triangular orientation with respect to the chromosome order (interpreting sides 1 and 2 as i and j axes in a matrix coordinate system).
2. Sorted: once the mates are oriented, the pair records are lexically sorted by chrom1, chrom2, pos1, pos2. With (1) and (2), contacts are said to be sorted by chromosome-chromosome block, a.k.a. "block" sorted.
3. Indexed: we use [bgzip](http://www.htslib.org/doc/tabix.html) to compress the file and [Pairix](https://github.com/4dn-dcic/pairix) to index it. This creates a small .px2 index file which facilitates 2-dimensional queries on the reads.

Notes:

* Having an indexed pairs file can also be useful of other kinds of read-level analyses.
* If (1) is already satisfied, you can also prepare a Pairix-indexed file manually, without cooler csort. See this [example](https://github.com/4dn-dcic/pairix#usage-examples-for-pairix)

### Sort and index

!cooler csort -h

Usage: cooler csort [OPTIONS] PAIRS\_PATH CHROMOSOMES\_PATH

Sort and index a contact list.

Order the mates of each pair record so that all contacts are upper

triangular with respect to the chromosome ordering given by the

chromosomes file, sort contacts by genomic location, and index the

resulting file.

Notes:

- csort can also be used to sort and index a text representation of

a contact \_matrix\_ in bedGraph-like format. In this case, substitute

`pos1` and `pos2` with `start1` and `start2`, respectively.

- Requires Unix tools: sort, bgzip + tabix or pairix.

If indexing with Tabix, the output file will have the following

properties:

- Upper triangular: the read pairs on each row are assigned to side 1 or 2

in such a way that (chrom1, pos1) is always "less than" (chrom2, pos2)

- Rows are lexicographically sorted by chrom1, pos1, chrom2, pos2;

i.e. "positionally sorted"

- Compressed with bgzip [\*]

- Indexed using Tabix [\*] on chrom1 and pos1.

If indexing with Pairix, the output file will have the following

properties:

- Upper triangular: the read pairs on each row are assigned to side 1 or 2

in such a way that (chrom1, pos1) is always "less than" (chrom2, pos2)

- Rows are lexicographically sorted by chrom1, chrom2, pos1, pos2; i.e.

"block sorted"

- Compressed with bgzip [\*]

- Indexed using Pairix [+] on chrom1, chrom2 and pos1.

[\*] Tabix manpage: <http://www.htslib.org/doc/tabix.html>.

[+] Pairix on Github: <https://github.com/4dn-dcic/pairix>

Arguments:

PAIRS\_PATH : Contacts (i.e. read pairs) text file, optionally compressed.

CHROMOSOMES\_PATH : File listing desired chromosomes in the desired order.

May be tab-delimited, e.g. a UCSC-like chromsizes file. Contacts mapping

to other chromosomes will be discarded.

Options:

-c1, --chrom1 INTEGER chrom1 field number in the input file (starting

from 1) [required]

-c2, --chrom2 INTEGER chrom2 field number [required]

-p1, --pos1 INTEGER pos1 field number [required]

-p2, --pos2 INTEGER pos2 field number [required]

-i, --index [tabix|pairix] Select the preset sort and indexing options

[default: pairix]

--flip-only Only flip mates; no sorting or indexing. Write

to stdout. [default: False]

-p, --nproc INTEGER Number of processors [default: 8]

-0, --zero-based Read positions are zero-based [default: False]

--sep TEXT Data delimiter in the input file [default: \t]

--comment-char TEXT Comment character to skip header [default: #]

--sort-options TEXT Quoted list of additional options to `sort`

command

-o, --out TEXT Output gzip file

-s1, --strand1 INTEGER strand1 field number (deprecated)

-s2, --strand2 INTEGER strand2 field number (deprecated)

-h, --help Show this message and exit.

%%bash

# Note that the input pairs file happens to be space-delimited, which we specify

# with the --sep argument (tab is assumed by default).

# The output pairs file will always be tab-delimited!

​

CHROMSIZES\_FILE='data/b37.chrom.sizes.reduced'

PAIRS\_FILE='data/GSM1551552\_HIC003\_merged\_nodups.txt.subset.gz'

​

cooler csort -c1 3 -p1 4 -c2 7 -p2 8 --sep ' ' --out pairs.sorted.txt.gz $PAIRS\_FILE $CHROMSIZES\_FILE

INFO:cooler:Enumerating requested chromosomes...

INFO:cooler:1 1

INFO:cooler:2 2

INFO:cooler:3 3

INFO:cooler:4 4

INFO:cooler:5 5

INFO:cooler:6 6

INFO:cooler:7 7

INFO:cooler:8 8

INFO:cooler:9 9

INFO:cooler:10 10

INFO:cooler:11 11

INFO:cooler:12 12

INFO:cooler:13 13

INFO:cooler:14 14

INFO:cooler:15 15

INFO:cooler:16 16

INFO:cooler:17 17

INFO:cooler:18 18

INFO:cooler:19 19

INFO:cooler:20 20

INFO:cooler:21 21

INFO:cooler:22 22

INFO:cooler:X 23

INFO:cooler:Y 24

INFO:cooler:MT 25

INFO:cooler:Input: 'data/GSM1551552\_HIC003\_merged\_nodups.txt.subset.gz'

INFO:cooler:Output: 'pairs.sorted.txt.gz'

INFO:cooler:Reordering pair mates and sorting pair records...

INFO:cooler:Sort order: block (chrom1, chrom2, pos1, pos2)

INFO:cooler:sort -k3,3 -k7,7 -k4,4n -k8,8n --parallel=8 --buffer-size=50%

INFO:cooler:Indexing...

INFO:cooler:Indexer: pairix

INFO:cooler:pairix -f -s3 -d7 -b4 -e4 -u8 -v8 pairs.sorted.txt.gz

%%bash

​

# What's in the output?

zcat pairs.sorted.txt.gz | head

D260LACXX130602:2:2315:7361:72358 0 1 85378 186 16 1 591085 1097 0 0

D258GACXX130605:8:2316:2958:10584 0 1 719104 1418 16 1 728403 1406 20 10

C24LCACXX130513:8:2315:5697:82732 0 1 758309 1523 0 1 43498676 121266 68 120

D260LACXX130602:2:2209:16327:23744 0 1 784407 527093 16 1 229918735 1593 88 40

D260LACXX130602:2:2202:16754:100485 16 1 890311 1801 16 1 993887 2056 165 18

D260LACXX130602:2:2309:8547:48542 0 1 925938 1866 16 1 1034493 2117 178 60

C24LCACXX130513:8:2312:14225:27548 0 1 941657 1904 0 1 1620964 3551 178 0

D258GACXX130605:8:1201:10692:67155 0 1 949416 342780 16 1 155311666 1935 178 178

C24LCACXX130513:8:2307:2896:10307 0 1 951652 3944 16 1 1751062 1946 146 175

D258GACXX130605:8:2308:5276:50416 0 1 992230 2049 0 1 1255504 2556 175 178

Finally, using cooler cload, we aggregate (bin) the contacts in pairs.sorted.txt.gz against the bins file, bins.1000kb.bed, and write the contents to the binary test.cool file.

### Load a pairix-indexed contact list

We now support the [4DN-DCIC's](https://github.com/4dn-dcic) [Pairix](https://github.com/4dn-dcic/pairix) standard for contact lists (i.e. pairs files).

A Pairix-indexed file has the advantage of 2D querying. However, it uses a slightly different sorting convention:

1. Like the previous Tabix scheme, interchromosomal pairs in Pairix files should consistently respect some order of the chromosomes (i.e. be "upper triangular"). Unlike the previous scheme, the chromosome order used to create the pairs file can be arbitrary, and does not need to match the order you wish to use in the cooler file.
2. Unlike the previous Tabix scheme, where the file is sorted by chrom1, pos1, chrom2, pos2, Pairix files are sorted by chrom1, chrom2, pos1, pos2.

%%bash

​

cooler cload pairix -h

Usage: cooler cload pairix [OPTIONS] BINS PAIRS\_PATH COOL\_PATH

Bin a pairix-indexed contact list file.

BINS : One of the following

<TEXT:INTEGER> : 1. Path to a chromsizes file, 2. Bin size in bp

<TEXT> : Path to BED file defining the genomic bin segmentation.

PAIRS\_PATH : Path to contacts (i.e. read pairs) file.

COOL\_PATH : Output COOL file path or URI.

See also: 'cooler csort' to sort and index a contact list file

Pairix on GitHub: <https://github.com/4dn-dcic/pairix>.

Options:

--metadata TEXT Path to JSON file containing user metadata.

--assembly TEXT Name of genome assembly (e.g. hg19, mm10)

-p, --nproc INTEGER Number of processes to split the work between.

[default: 8]

-0, --zero-based Positions are zero-based [default: False]

-s, --max-split INTEGER Divide the pairs from each chromosome into at most

this many chunks. Smaller chromosomes will be split

less frequently or not at all. Increase ths value

if large chromosomes dominate the workload on

multiple processors. [default: 2]

-h, --help Show this message and exit.

%%bash

​

# alternatively, we could pass $CHROMSIZES\_FILE:1000000 below instead of creating $BINS\_FILE

BINS\_FILE='bins.1000kb.bed'

INDEXED\_PAIRS\_FILE='pairs.sorted.txt.gz'

OUTPUT\_FILE='test.cool'

​

cooler cload pairix $BINS\_FILE $INDEXED\_PAIRS\_FILE $OUTPUT\_FILE

INFO:cooler:Using 8 cores

INFO:cooler:Creating cooler at "test.cool::/"

INFO:cooler:Writing chroms

INFO:cooler:Writing bins

INFO:cooler:Writing pixels

INFO:cooler:Binning 1:0-125000000|\*

INFO:cooler:Binning 1:125000000-249250621|\*

INFO:cooler:Binning 2:0-129000000|\*

INFO:cooler:Binning 2:129000000-243199373|\*

INFO:cooler:Binning 3:0-158000000|\*

INFO:cooler:Binning 3:158000000-198022430|\*

INFO:cooler:Binning 4:0-163000000|\*

INFO:cooler:Binning 4:163000000-191154276|\*

INFO:cooler:Finished 4:163000000-191154276|\*

INFO:cooler:Binning 5:0-173000000|\*

INFO:cooler:Finished 3:158000000-198022430|\*

INFO:cooler:Binning 5:173000000-180915260|\*

INFO:cooler:Finished 5:173000000-180915260|\*

INFO:cooler:Binning 6:0-171115067|\*

INFO:cooler:Finished 2:129000000-243199373|\*

INFO:cooler:Binning 7:0-159138663|\*

INFO:cooler:Finished 1:125000000-249250621|\*

INFO:cooler:Binning 8:0-146364022|\*

INFO:cooler:Finished 1:0-125000000|\*

INFO:cooler:Finished 2:0-129000000|\*

INFO:cooler:Binning 9:0-141213431|\*

INFO:cooler:Binning 10:0-135534747|\*

INFO:cooler:Finished 4:0-163000000|\*

INFO:cooler:Binning 11:0-135006516|\*

INFO:cooler:Finished 3:0-158000000|\*

INFO:cooler:Binning 12:0-133851895|\*

INFO:cooler:Finished 5:0-173000000|\*

INFO:cooler:Binning 13:0-115169878|\*

INFO:cooler:Finished 6:0-171115067|\*

INFO:cooler:Binning 14:0-107349540|\*

INFO:cooler:Finished 10:0-135534747|\*

INFO:cooler:Finished 9:0-141213431|\*

INFO:cooler:Finished 12:0-133851895|\*

INFO:cooler:Finished 11:0-135006516|\*

INFO:cooler:Binning 15:0-102531392|\*

INFO:cooler:Binning 16:0-90354753|\*

INFO:cooler:Binning 17:0-81195210|\*

INFO:cooler:Binning 18:0-78077248|\*

INFO:cooler:Finished 8:0-146364022|\*

INFO:cooler:Finished 7:0-159138663|\*

INFO:cooler:Finished 13:0-115169878|\*

INFO:cooler:Binning 19:0-59128983|\*

INFO:cooler:Binning 20:0-63025520|\*

INFO:cooler:Binning 21:0-48129895|\*

INFO:cooler:Finished 14:0-107349540|\*

INFO:cooler:Binning 22:0-51304566|\*

INFO:cooler:Finished 21:0-48129895|\*

INFO:cooler:Binning X:0-155270560|\*

INFO:cooler:Finished 19:0-59128983|\*

INFO:cooler:Binning Y:0-59373566|\*

INFO:cooler:Finished 20:0-63025520|\*

INFO:cooler:Finished 22:0-51304566|\*

INFO:cooler:Finished 18:0-78077248|\*

INFO:cooler:Binning MT:0-16569|\*

INFO:cooler:Finished 17:0-81195210|\*

INFO:cooler:Finished Y:0-59373566|\*

INFO:cooler:Finished MT:0-16569|\*

INFO:cooler:Finished 16:0-90354753|\*

INFO:cooler:Finished 15:0-102531392|\*

INFO:cooler:Finished X:0-155270560|\*

INFO:cooler:Writing indexes

INFO:cooler:Writing info

INFO:cooler:Done

# Cooler Python API

Pre-requisites:

* Basic Python knowledge
* Some experience with the NumPy array package (or MATLAB)

This walkthrough also makes use of:

* Jupyter (IPython) Notebook. That's what you're using right now!
* Pandas, the dataframe package (similar to R data.frames)
* h5py, the package to interact with HDF5 files from Python
* matplotlib, the MATLAB-inspired plotting package for Python

To navigate this notebook:

* Click on a code cell and execute its code by pressing shift+enter or clicking the "play" button on the toolbar.
* While the code cell is running the prompt on the left will look like In [\*]: and will display the execution count when it is done.
* Execution output will be displayed beneath each cell.
* To restart the notebook, use the options in the Kernel dropdown menu.
* You can also run the entire notebook in one go with the Restart & Run All option

# Import the packages we will use

import matplotlib.pyplot as plt

import numpy as np

import pandas

import h5py

​

import cooler

# The following directive activates inline plotting

%matplotlib inline

filepath = 'data/Rao2014-GM12878-MboI-allreps-filtered.5kb.cool'

## Direct access with h5py

The h5py library (HDF5 for Python) provides an excellent Pythonic interface between HDF5 and native [NumPy](http://www.numpy.org/) arrays and dtypes. It allows you to treat an HDF5 file like a dictionary with complete access to the file's contents as well as the ability to manipulate groups and read or write datasets and attributes. There is additionally a low-level API that wraps the libhdf5 C functions directly. See the [h5py docs](http://docs.h5py.org/en/latest/index.html).

h5 = h5py.File(filepath, 'r')

h5

<HDF5 file "Rao2014-GM12878-MboI-allreps-filtered.5kb.cool" (mode r)>

h5.keys()

KeysView(<HDF5 file "Rao2014-GM12878-MboI-allreps-filtered.5kb.cool" (mode r)>)

Files and Groups are dict-like.

h5['pixels']

<HDF5 group "/pixels" (3 members)>

list(h5['pixels'].keys())

['bin1\_id', 'bin2\_id', 'count']

h5py dataset objects are views onto the data on disk

h5['pixels']['bin2\_id']

<HDF5 dataset "bin2\_id": shape (1543535265,), type "<i8">

Slicing or indexing returns a numpy array in memory.

h5['pixels']['bin2\_id'][:10]

array([ 234, 1994, 3258, 4087, 6093, 37359, 49826, 49889, 58451, 60826])

h5['pixels']['count'][:10]

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

h5.close()

The Python cooler package is just a thin wrapper over h5py.

* It lets you access the data tables as [Pandas](http://pandas.pydata.org/) [data frames and series](http://pandas.pydata.org/pandas-docs/stable/10min.html).
* It also provides a matrix abstraction: letting you query the upper triangle pixel table as if it were a full rectangular [sparse matrix](http://www.scipy-lectures.org/advanced/scipy_sparse/storage_schemes.html) via [SciPy](http://www.scipy-lectures.org/index.html).

See below.

## The Cooler class

Accepts a file path or an open HDF5 file object.

NOTE: Using a filepath allows the Cooler object to be serialized/pickled since the file is only opened when needed.

c = cooler.Cooler(filepath)

### The info dictionary

c.info

{'bin-size': 5000,

'bin-type': 'fixed',

'creation-date': '2016-02-25T22:53:09.510744',

'format-url': 'https://github.com/mirnylab/cooler',

'format-version': 2,

'genome-assembly': 'hg19',

'id': None,

'library-version': '0.3.0',

'metadata': {'QC': {'double-sided': {'filtered-invalid': {'removed-dangling-ends': 234897003,

'removed-error-pair': 6074295,

'removed-self-circles': 1741768},

'filtered-valid': {'removed-duplicate': 110650005,

'removed-large-small-pair': 657466,

'removed-outlier-fragment': 151337031,

'removed-start-near-rsite': ''},

'total': 3390352656,

'valid': 3147639590},

'post-filtering': {'cis': 2085711027,

'total': 2884995088,

'trans': 799284061},

'pre-filtering': {'double-sided': 3390352656,

'single-sided': 1942368995,

'total': 5332721651,

'unused': 0}},

'cell-type': 'GM12878',

'enzyme': 'MboI',

'publication': '',

'sex': 'F',

'species': 'Homo sapiens'},

'nbins': 619150,

'nchroms': 25,

'nnz': 1543535265}

### Table Views

Tables are accessed via methods.

c.chroms()

<cooler.core.RangeSelector1D at 0x7fa09501d940>

The return value is a selector or "view" on a table that accepts column and range queries ("slices").

* Column selections return a new view.
* Range selections return pandas [DataFrames or Series](http://pandas.pydata.org/pandas-docs/stable/dsintro.html).

c.chroms()[1:5]

|  | name | length |
| --- | --- | --- |
| 1 | chr2 | 243199373 |
| 2 | chr3 | 198022430 |
| 3 | chr4 | 191154276 |
| 4 | chr5 | 180915260 |

# get the whole table

c.chroms()[:]

|  | name | length |
| --- | --- | --- |
| 0 | chr1 | 249250621 |
| 1 | chr2 | 243199373 |
| 2 | chr3 | 198022430 |
| 3 | chr4 | 191154276 |
| 4 | chr5 | 180915260 |
| 5 | chr6 | 171115067 |
| 6 | chr7 | 159138663 |
| 7 | chr8 | 146364022 |
| 8 | chr9 | 141213431 |
| 9 | chr10 | 135534747 |
| 10 | chr11 | 135006516 |
| 11 | chr12 | 133851895 |
| 12 | chr13 | 115169878 |
| 13 | chr14 | 107349540 |
| 14 | chr15 | 102531392 |
| 15 | chr16 | 90354753 |
| 16 | chr17 | 81195210 |
| 17 | chr18 | 78077248 |
| 18 | chr19 | 59128983 |
| 19 | chr20 | 63025520 |
| 20 | chr21 | 48129895 |
| 21 | chr22 | 51304566 |
| 22 | chrX | 155270560 |
| 23 | chrY | 59373566 |
| 24 | chrM | 16571 |

# more convenient access to chromosomes

c.chromnames

['chr1',

'chr2',

'chr3',

'chr4',

'chr5',

'chr6',

'chr7',

'chr8',

'chr9',

'chr10',

'chr11',

'chr12',

'chr13',

'chr14',

'chr15',

'chr16',

'chr17',

'chr18',

'chr19',

'chr20',

'chr21',

'chr22',

'chrX',

'chrY',

'chrM']

In the bin table, the weight column contains the matrix balancing weights computed for each genomic bin.

c.bins()[:10]

|  | chrom | start | end | weight |
| --- | --- | --- | --- | --- |
| 0 | chr1 | 0 | 5000 | NaN |
| 1 | chr1 | 5000 | 10000 | NaN |
| 2 | chr1 | 10000 | 15000 | NaN |
| 3 | chr1 | 15000 | 20000 | NaN |
| 4 | chr1 | 20000 | 25000 | NaN |
| 5 | chr1 | 25000 | 30000 | NaN |
| 6 | chr1 | 30000 | 35000 | NaN |
| 7 | chr1 | 35000 | 40000 | NaN |
| 8 | chr1 | 40000 | 45000 | NaN |
| 9 | chr1 | 45000 | 50000 | NaN |

Selecting a list of columns returns a new DataFrame view on that subset of columns

bins = c.bins()[['chrom', 'start', 'end']]

bins

<cooler.core.RangeSelector1D at 0x7fa094580dd8>

bins[:10]

|  | chrom | start | end |
| --- | --- | --- | --- |
| 0 | chr1 | 0 | 5000 |
| 1 | chr1 | 5000 | 10000 |
| 2 | chr1 | 10000 | 15000 |
| 3 | chr1 | 15000 | 20000 |
| 4 | chr1 | 20000 | 25000 |
| 5 | chr1 | 25000 | 30000 |
| 6 | chr1 | 30000 | 35000 |
| 7 | chr1 | 35000 | 40000 |
| 8 | chr1 | 40000 | 45000 |
| 9 | chr1 | 45000 | 50000 |

Selecting a single column returns a Series view

weights = c.bins()['weight']

weights

<cooler.core.RangeSelector1D at 0x7fa09459b5c0>

weights[500:510]

500 1.021570

501 1.935517

502 0.584862

503 0.895666

504 1.302236

505 0.845886

506 0.992397

507 1.470796

508 1.022486

509 0.952231

Name: weight, dtype: float64

The pixel table contains the non-zero upper triangle entries of the contact map.

c.pixels()[:10]

|  | bin1\_id | bin2\_id | count |
| --- | --- | --- | --- |
| 0 | 2 | 234 | 1 |
| 1 | 2 | 1994 | 1 |
| 2 | 2 | 3258 | 1 |
| 3 | 2 | 4087 | 1 |
| 4 | 2 | 6093 | 1 |
| 5 | 2 | 37359 | 1 |
| 6 | 2 | 49826 | 1 |
| 7 | 2 | 49889 | 1 |
| 8 | 2 | 58451 | 1 |
| 9 | 2 | 60826 | 1 |

Use the join=True option if you would like to expand the bin IDs into genomic bin coordinates by joining the output with the bin table.

c.pixels(join=True)[:10]

|  | chrom1 | start1 | end1 | chrom2 | start2 | end2 | count |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | chr1 | 10000 | 15000 | chr1 | 1170000 | 1175000 | 1 |
| 1 | chr1 | 10000 | 15000 | chr1 | 9970000 | 9975000 | 1 |
| 2 | chr1 | 10000 | 15000 | chr1 | 16290000 | 16295000 | 1 |
| 3 | chr1 | 10000 | 15000 | chr1 | 20435000 | 20440000 | 1 |
| 4 | chr1 | 10000 | 15000 | chr1 | 30465000 | 30470000 | 1 |
| 5 | chr1 | 10000 | 15000 | chr1 | 186795000 | 186800000 | 1 |
| 6 | chr1 | 10000 | 15000 | chr1 | 249130000 | 249135000 | 1 |
| 7 | chr1 | 10000 | 15000 | chr2 | 190000 | 195000 | 1 |
| 8 | chr1 | 10000 | 15000 | chr2 | 43000000 | 43005000 | 1 |
| 9 | chr1 | 10000 | 15000 | chr2 | 54875000 | 54880000 | 1 |

Pandas lets you readily dump any table selection to tabular text file.

df = c.pixels(join=True)[:100]

​

# tab-delimited file, don't write the index column or header row

df.to\_csv('myselection.txt', sep='\t', index=False, header=False)

!head myselection.txt

chr1 10000 15000 chr1 1170000 1175000 1

chr1 10000 15000 chr1 9970000 9975000 1

chr1 10000 15000 chr1 16290000 16295000 1

chr1 10000 15000 chr1 20435000 20440000 1

chr1 10000 15000 chr1 30465000 30470000 1

chr1 10000 15000 chr1 186795000 186800000 1

chr1 10000 15000 chr1 249130000 249135000 1

chr1 10000 15000 chr2 190000 195000 1

chr1 10000 15000 chr2 43000000 43005000 1

chr1 10000 15000 chr2 54875000 54880000 1

### Bin annotation

Another way to annotate the bins in a data frame of pixels is to use cooler.annotate. It does a [left outer join](http://chris.friedline.net/2015-12-15-rutgers/lessons/python2/04-merging-data.html) from the bin1\_id and bin2\_id columns onto a data frame indexed by bin ID that describes the bins.

bins = c.bins()[:] # fetch all the bins

​

pix = c.pixels()[100:110] # select some pixels with unannotated bins

pix

|  | bin1\_id | bin2\_id | count |
| --- | --- | --- | --- |
| 100 | 2 | 356649 | 1 |
| 101 | 2 | 356700 | 1 |
| 102 | 2 | 363068 | 1 |
| 103 | 2 | 363243 | 1 |
| 104 | 2 | 363270 | 1 |
| 105 | 2 | 363293 | 1 |
| 106 | 2 | 363329 | 1 |
| 107 | 2 | 363348 | 1 |
| 108 | 2 | 363373 | 1 |
| 109 | 2 | 363374 | 1 |

cooler.annotate(pix, bins)

|  | chrom1 | start1 | end1 | weight1 | chrom2 | start2 | end2 | weight2 | count |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | chr1 | 10000 | 15000 | NaN | chr10 | 102850000 | 102855000 | 1.503567 | 1 |
| 101 | chr1 | 10000 | 15000 | NaN | chr10 | 103105000 | 103110000 | 0.924921 | 1 |
| 102 | chr1 | 10000 | 15000 | NaN | chr10 | 134945000 | 134950000 | 1.480836 | 1 |
| 103 | chr1 | 10000 | 15000 | NaN | chr11 | 285000 | 290000 | 1.455694 | 1 |
| 104 | chr1 | 10000 | 15000 | NaN | chr11 | 420000 | 425000 | 1.107395 | 1 |
| 105 | chr1 | 10000 | 15000 | NaN | chr11 | 535000 | 540000 | 0.819330 | 1 |
| 106 | chr1 | 10000 | 15000 | NaN | chr11 | 715000 | 720000 | 0.911051 | 1 |
| 107 | chr1 | 10000 | 15000 | NaN | chr11 | 810000 | 815000 | 1.202326 | 1 |
| 108 | chr1 | 10000 | 15000 | NaN | chr11 | 935000 | 940000 | 0.889250 | 1 |
| 109 | chr1 | 10000 | 15000 | NaN | chr11 | 940000 | 945000 | 1.017561 | 1 |

cooler.annotate(pix, bins[['weight']], replace=False)

|  | weight1 | weight2 | bin1\_id | bin2\_id | count |
| --- | --- | --- | --- | --- | --- |
| 100 | NaN | 1.503567 | 2 | 356649 | 1 |
| 101 | NaN | 0.924921 | 2 | 356700 | 1 |
| 102 | NaN | 1.480836 | 2 | 363068 | 1 |
| 103 | NaN | 1.455694 | 2 | 363243 | 1 |
| 104 | NaN | 1.107395 | 2 | 363270 | 1 |
| 105 | NaN | 0.819330 | 2 | 363293 | 1 |
| 106 | NaN | 0.911051 | 2 | 363329 | 1 |
| 107 | NaN | 1.202326 | 2 | 363348 | 1 |
| 108 | NaN | 0.889250 | 2 | 363373 | 1 |
| 109 | NaN | 1.017561 | 2 | 363374 | 1 |

### Enter The Matrix

Finally, the matrix method provides a 2D-sliceable view on the data. It allows you to query the data on file as a full rectangular contact matrix.

c.matrix()

<cooler.core.RangeSelector2D at 0x7fa093bba080>

The result of a query is a 2D NumPy array.

arr = c.matrix(balance=False)[1000:1200, 1000:1200]

arr

array([[129, 230, 72, ..., 3, 3, 3],

[230, 217, 128, ..., 3, 6, 1],

[ 72, 128, 42, ..., 3, 1, 1],

...,

[ 3, 3, 3, ..., 199, 357, 165],

[ 3, 6, 1, ..., 357, 225, 257],

[ 3, 1, 1, ..., 165, 257, 136]])

Use sparse=True to return scipy.sparse.coo\_matrix objects instead.

mat = c.matrix(balance=False, sparse=True)[1000:1200, 1000:1200]

mat

<200x200 sparse matrix of type '<class 'numpy.int64'>'

with 39393 stored elements in COOrdinate format>

It is straightforward to convert to a dense 2D numpy array.

arr = mat.toarray()

arr

array([[129, 230, 72, ..., 3, 3, 3],

[230, 217, 128, ..., 3, 6, 1],

[ 72, 128, 42, ..., 3, 1, 1],

...,

[ 3, 3, 3, ..., 199, 357, 165],

[ 3, 6, 1, ..., 357, 225, 257],

[ 3, 1, 1, ..., 165, 257, 136]])

Notice that the lower triangle has been automatically filled in.

fig = plt.figure(figsize=(10, 10))

ax = fig.add\_subplot(111)

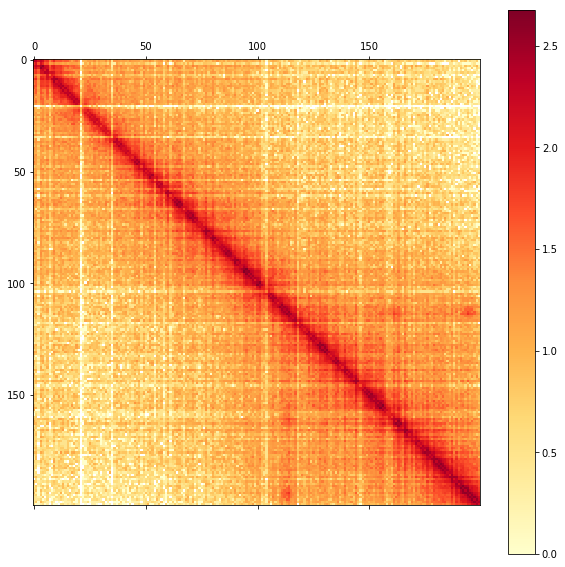
im = ax.matshow(np.log10(arr), cmap='YlOrRd')

fig.colorbar(im)

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:3: RuntimeWarning: divide by zero encountered in log10

This is separate from the ipykernel package so we can avoid doing imports until

<matplotlib.colorbar.Colorbar at 0x7fa094234240>



Notice the light and dark "banded" appearance? That's because you are looking at the unnormalized counts.

### Balancing your selection

We usually normalize or "correct" Hi-C using a technique called matrix balancing. This involves finding a set of weights or biases 𝑏𝑖

for each bin 𝑖

such that

𝑁𝑜𝑟𝑚𝑎𝑙𝑖𝑧𝑒𝑑[𝑖,𝑗]=𝑂𝑏𝑠𝑒𝑟𝑣𝑒𝑑[𝑖,𝑗]⋅𝑏[𝑖]⋅𝑏[𝑗],

such that the marginals (i.e., row/column sums) of the global contact matrix are flat and equal.

Cooler can store the pre-computed balancing weights in the bin table.

Here's one way to manually apply them to balance your selection.

# get the balancing weights as a numpy array

weights = c.bins()['weight'] # view

bias = weights[1000:1200] # series

bias = bias.values # array

​

# fetch a sparse matrix

mat = c.matrix(balance=False, sparse=True)[1000:1200, 1000:1200]

​

# apply the balancing weights

mat.data = bias[mat.row] \* bias[mat.col] \* mat.data

​

# convert to dense numpy array

arr = mat.toarray()

As a shortcut, we get the same result by passing balance=True to the matrix view constructor.

arr2 = c.matrix(balance=True, sparse=True)[1000:1200, 1000:1200].toarray()

np.allclose(arr, arr2, equal\_nan=True)

True

fig = plt.figure(figsize=(10, 10))

ax = fig.add\_subplot(111)

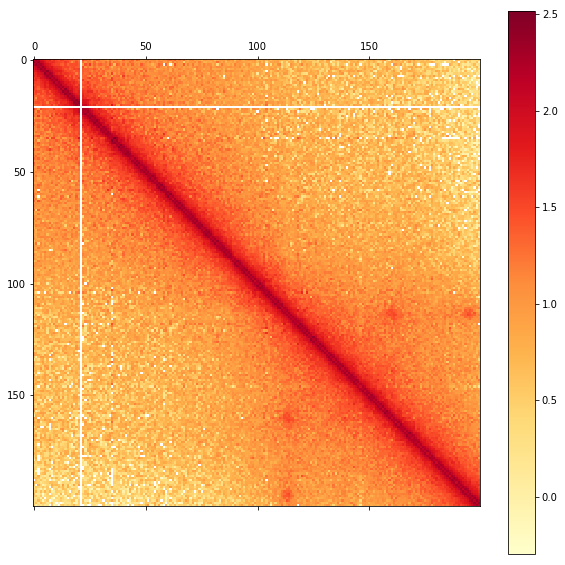
im = ax.matshow(np.log10(arr), cmap='YlOrRd')

fig.colorbar(im)

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:3: RuntimeWarning: divide by zero encountered in log10

This is separate from the ipykernel package so we can avoid doing imports until

<matplotlib.colorbar.Colorbar at 0x7fa0940eff98>



### Genomic coordinate range selection

The bin table, pixel table and matrix views also accept UCSC-style genomic range strings or (chrom, start, end) triples.

c.bins().fetch('chr2:10,000,000-20,000,000')

|  | chrom | start | end | weight |
| --- | --- | --- | --- | --- |
| 51851 | chr2 | 10000000 | 10005000 | 2.474065 |
| 51852 | chr2 | 10005000 | 10010000 | 1.403127 |
| 51853 | chr2 | 10010000 | 10015000 | 0.883045 |
| 51854 | chr2 | 10015000 | 10020000 | 0.771122 |
| 51855 | chr2 | 10020000 | 10025000 | 0.924510 |
| 51856 | chr2 | 10025000 | 10030000 | 0.866517 |
| 51857 | chr2 | 10030000 | 10035000 | 1.014915 |
| 51858 | chr2 | 10035000 | 10040000 | 1.052058 |
| 51859 | chr2 | 10040000 | 10045000 | 1.311514 |
| 51860 | chr2 | 10045000 | 10050000 | 0.965850 |
| 51861 | chr2 | 10050000 | 10055000 | 0.613818 |
| 51862 | chr2 | 10055000 | 10060000 | 1.002511 |
| 51863 | chr2 | 10060000 | 10065000 | 1.024848 |
| 51864 | chr2 | 10065000 | 10070000 | 2.008198 |
| 51865 | chr2 | 10070000 | 10075000 | 1.919790 |
| 51866 | chr2 | 10075000 | 10080000 | 0.880975 |
| 51867 | chr2 | 10080000 | 10085000 | 0.856335 |
| 51868 | chr2 | 10085000 | 10090000 | 0.728885 |
| 51869 | chr2 | 10090000 | 10095000 | 1.206524 |
| 51870 | chr2 | 10095000 | 10100000 | 0.839497 |
| 51871 | chr2 | 10100000 | 10105000 | 1.080518 |
| 51872 | chr2 | 10105000 | 10110000 | 0.638659 |
| 51873 | chr2 | 10110000 | 10115000 | 1.376984 |
| 51874 | chr2 | 10115000 | 10120000 | 0.927573 |
| 51875 | chr2 | 10120000 | 10125000 | 0.701900 |
| 51876 | chr2 | 10125000 | 10130000 | 0.669814 |
| 51877 | chr2 | 10130000 | 10135000 | 0.722137 |
| 51878 | chr2 | 10135000 | 10140000 | 0.549518 |
| 51879 | chr2 | 10140000 | 10145000 | 0.670836 |
| 51880 | chr2 | 10145000 | 10150000 | 0.817080 |
| ... | ... | ... | ... | ... |
| 53821 | chr2 | 19850000 | 19855000 | 0.951929 |
| 53822 | chr2 | 19855000 | 19860000 | 1.174055 |
| 53823 | chr2 | 19860000 | 19865000 | 0.971085 |
| 53824 | chr2 | 19865000 | 19870000 | 1.331585 |
| 53825 | chr2 | 19870000 | 19875000 | 0.704319 |
| 53826 | chr2 | 19875000 | 19880000 | 0.821776 |
| 53827 | chr2 | 19880000 | 19885000 | 0.664968 |
| 53828 | chr2 | 19885000 | 19890000 | 0.696820 |
| 53829 | chr2 | 19890000 | 19895000 | 0.767861 |
| 53830 | chr2 | 19895000 | 19900000 | 0.852783 |
| 53831 | chr2 | 19900000 | 19905000 | 0.623329 |
| 53832 | chr2 | 19905000 | 19910000 | 0.841017 |
| 53833 | chr2 | 19910000 | 19915000 | 0.845765 |
| 53834 | chr2 | 19915000 | 19920000 | 0.953318 |
| 53835 | chr2 | 19920000 | 19925000 | 1.668928 |
| 53836 | chr2 | 19925000 | 19930000 | 0.887702 |
| 53837 | chr2 | 19930000 | 19935000 | 1.129732 |
| 53838 | chr2 | 19935000 | 19940000 | 0.753862 |
| 53839 | chr2 | 19940000 | 19945000 | 1.566019 |
| 53840 | chr2 | 19945000 | 19950000 | 1.471512 |
| 53841 | chr2 | 19950000 | 19955000 | 0.507452 |
| 53842 | chr2 | 19955000 | 19960000 | 0.918652 |
| 53843 | chr2 | 19960000 | 19965000 | 1.055917 |
| 53844 | chr2 | 19965000 | 19970000 | 0.989101 |
| 53845 | chr2 | 19970000 | 19975000 | 0.923316 |
| 53846 | chr2 | 19975000 | 19980000 | 0.575194 |
| 53847 | chr2 | 19980000 | 19985000 | 1.450563 |
| 53848 | chr2 | 19985000 | 19990000 | 0.861964 |
| 53849 | chr2 | 19990000 | 19995000 | 0.786807 |
| 53850 | chr2 | 19995000 | 20000000 | 0.692477 |

2000 rows × 4 columns

cis = c.matrix(sparse=True).fetch('chr21')

cis.shape

(9626, 9626)

trans = c.matrix(sparse=True).fetch('chr21', 'chr22')

trans.shape

(9626, 10261)

## Function API

Instead of the methods of the Cooler class, you can use the similarly named functions in the cooler module directly. However, they will only accept an open HDF5 file handle, not a file path string, and they execute their queries eagerly.

Open the HDF5 file with h5py

h5 = h5py.File(filepath, 'r')

cooler.info(h5)

{'bin-size': 5000,

'bin-type': 'fixed',

'creation-date': '2016-02-25T22:53:09.510744',

'format-url': 'https://github.com/mirnylab/cooler',

'format-version': 2,

'genome-assembly': 'hg19',

'id': None,

'library-version': '0.3.0',

'metadata': {'QC': {'double-sided': {'filtered-invalid': {'removed-dangling-ends': 234897003,

'removed-error-pair': 6074295,

'removed-self-circles': 1741768},

'filtered-valid': {'removed-duplicate': 110650005,

'removed-large-small-pair': 657466,

'removed-outlier-fragment': 151337031,

'removed-start-near-rsite': ''},

'total': 3390352656,

'valid': 3147639590},

'post-filtering': {'cis': 2085711027,

'total': 2884995088,

'trans': 799284061},

'pre-filtering': {'double-sided': 3390352656,

'single-sided': 1942368995,

'total': 5332721651,

'unused': 0}},

'cell-type': 'GM12878',

'enzyme': 'MboI',

'publication': '',

'sex': 'F',

'species': 'Homo sapiens'},

'nbins': 619150,

'nchroms': 25,

'nnz': 1543535265}

cooler.bins(h5, 0, 10)

|  | chrom | start | end | weight |
| --- | --- | --- | --- | --- |
| 0 | chr1 | 0 | 5000 | NaN |
| 1 | chr1 | 5000 | 10000 | NaN |
| 2 | chr1 | 10000 | 15000 | NaN |
| 3 | chr1 | 15000 | 20000 | NaN |
| 4 | chr1 | 20000 | 25000 | NaN |
| 5 | chr1 | 25000 | 30000 | NaN |
| 6 | chr1 | 30000 | 35000 | NaN |
| 7 | chr1 | 35000 | 40000 | NaN |
| 8 | chr1 | 40000 | 45000 | NaN |
| 9 | chr1 | 45000 | 50000 | NaN |

... etc.

Note that cooler.get() is a very generic utility that lets you interpret a HDF5 group containing 1D datasets as a table.

print(cooler.get.\_\_doc\_\_)

Query a range of rows from a table as a dataframe.

A table is an HDF5 group containing equal-length 1D datasets serving as

columns.

Parameters

----------

grp : ``h5py.Group`` or any dict-like of array-likes

Handle to an HDF5 group containing only 1D datasets or any similar

collection of 1D datasets or arrays

lo, hi : int, optional

Range of rows to select from the table.

fields : str or sequence of str, optional

Column or list of columns to query. Defaults to all available columns.

A single string returns a Series instead of a DataFrame.

convert\_enum : bool, optional

Whether to convert HDF5 enum datasets into ``pandas.Categorical``

columns instead of plain integer columns. Default is True.

kwargs : optional

Options to pass to ``pandas.DataFrame`` or ``pandas.Series``.

Returns

-------

DataFrame or Series

Notes

-----

HDF5 ASCII datasets are converted to Unicode.

# Raw P(s) curves by strand orientation

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

import multiprocess as mp

import numpy as np

import pandas as pd

import pypairix

import bioframe

import cooltools

mm9 = bioframe.fetch\_chromsizes('mm9')

chromsizes = bioframe.fetch\_chromsizes('mm9')

chromosomes = list(chromsizes.index)

conditions = ['WT', 'dN']

binsize = 100000

​

pairs\_paths = {

'WT': 'data/UNTR\_R1.nodups.pairs.gz',

'dN': 'data/NIPBL\_R1.nodups.pairs.gz'

}

​

long\_names = {

'WT': 'Wildtype',

'dN': 'NipblKO',

}

​

pal = sns.color\_palette('colorblind')

colors = {

'WT': pal[0],

'dN': pal[2],

}

from cooltools.expected import compute\_scaling

import bioframe.dask

pairs\_header = ['read\_id', 'chrom1', 'pos1', 'chrom2', 'pos2', 'strand1', 'strand2', 'pair\_type']

chrom = 'chr1'

​

df = bioframe.dask.read\_pairix\_block(

pairs\_paths['WT'],

(chrom, chrom),

names=pairs\_header,

chunk\_level=0)

df = df.compute()

print(len(df))

df.head()

4893134

|  | read\_id | chrom1 | pos1 | chrom2 | pos2 | strand1 | strand2 | pair\_type |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | . | chr1 | 3000031 | chr1 | 3002463 | - | + | LL |
| 1 | . | chr1 | 3000031 | chr1 | 64746728 | - | + | LL |
| 2 | . | chr1 | 3000036 | chr1 | 12271809 | - | + | LL |
| 3 | . | chr1 | 3000042 | chr1 | 3009611 | - | + | LL |
| 4 | . | chr1 | 3000048 | chr1 | 4637504 | - | + | LL |

plt.figure(figsize=(12, 12))

​

dmin = 10

dmax = int(1e7)

​

orientations = [

('+', '-'),

('-', '+'),

('+', '+'),

('-', '-')

]

​

grouped = df.groupby(['strand1', 'strand2'])

​

for s1, s2 in orientations:

group = grouped.get\_group((s1, s2))

dbins, obs, areas = compute\_scaling(

group,

region1=[0, chromsizes[chrom]],

region2=[0, chromsizes[chrom]],

dmin=dmin,

dmax=dmax)

dmeans = np.sqrt(dbins[:-1] \* dbins[1:])

scaling = obs / areas

plt.loglog(dmeans, scaling, label='{} {}'.format(s1, s2))

​

plt.xlabel(r'genomic separation $s$, bp')

plt.ylabel('pair frequency')

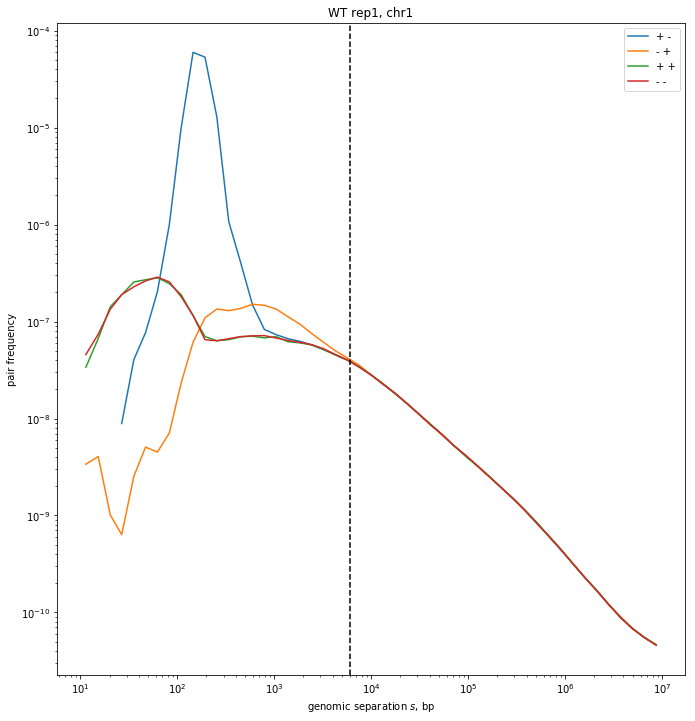
plt.legend()

plt.axvline(6000, c='k', ls='--')

plt.gca().set\_aspect(1)

plt.title(f'WT rep1, {chrom}')

<matplotlib.text.Text at 0x7feace921be0>



# P(s) on binned and balanced contact matrices, a.k.a. "expected"

%matplotlib inline

import matplotlib.pyplot as plt

import matplotlib as mpl

import seaborn as sns

import multiprocess as mp

import numpy as np

import pandas as pd

import bioframe

import cooltools

import cooler

mm9 = bioframe.fetch\_chromsizes('mm9')

chromsizes = bioframe.fetch\_chromsizes('mm9')

chromosomes = list(chromsizes.index)

conditions = ['WT', 'T', 'dN']

binsize = 100000

​

cooler\_paths = {

'WT' : f'data/UNTR.{binsize}.cool',

'T' : f'data/TAM.{binsize}.cool',

'dN' : f'data/NIPBL.{binsize}.cool',

}

long\_names = {

'WT': 'Wildtype',

'T' : 'TAM',

'dN': 'NipblKO',

}

pal = sns.color\_palette('colorblind')

colors = {

'WT': pal[0],

'T' : '#333333',

'dN': pal[2],

}

​

clrs = {

cond: cooler.Cooler(cooler\_paths[cond]) for cond in conditions

}

# this cell takes a long time to run

from cooltools.expected import diagsum, blocksum\_pairwise

supports = [(chrom, 0, chromsizes[chrom]) for chrom in chromosomes]

​

​

cis\_exp = {}

trs\_exp = {}

​

with mp.Pool() as pool:

for cond in conditions:

print(cond, 'cis')

tables = diagsum(

clrs[cond],

supports,

transforms={

'balanced': lambda p: p['count'] \* p['weight1'] \* p['weight2'],

},

chunksize=10000000,

ignore\_diags=2,

map=pool.map)

cis\_exp[cond] = pd.concat(

[tables[support] for support in supports],

keys=[support[0] for support in supports],

names=['chrom'])

cis\_exp[cond]['balanced.avg'] = cis\_exp[cond]['balanced.sum'] / cis\_exp[cond]['n\_valid']

cis\_exp[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.expected.cis.tsv', sep='\t')

print(cond, 'trans')

records = blocksum\_pairwise(

clrs[cond],

supports,

transforms={

'balanced': lambda p: p['count'] \* p['weight1'] \* p['weight2'],

},

chunksize=10000000,

map=pool.map)

trs\_exp[cond] = pd.DataFrame(

[{'chrom1': s1[0], 'chrom2': s2[0], \*\*rec} for (s1, s2), rec in records.items()],

columns=['chrom1', 'chrom2', 'n\_valid', 'count.sum', 'balanced.sum'])

trs\_exp[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.expected.trans.tsv', sep='\t', index=False)

WT cis

WT trans

T cis

T trans

dN cis

dN trans

cis\_exp['WT'].head()

|  |  | n\_valid | count.sum | balanced.sum | balanced.avg |
| --- | --- | --- | --- | --- | --- |
| chrom | diag |  |  |  |  |
| chr1 | 0 | 18900 | NaN | NaN | NaN |
| 1 | 18652 | NaN | NaN | NaN |
| 2 | 18638 | 132754.0 | 301.561046 | 0.016180 |
| 3 | 18634 | 92489.0 | 208.895766 | 0.011210 |
| 4 | 18623 | 70473.0 | 159.212659 | 0.008549 |

trs\_exp['WT'].head()

|  | chrom1 | chrom2 | n\_valid | count.sum | balanced.sum |
| --- | --- | --- | --- | --- | --- |
| 0 | chr1 | chr2 | 357714000 | 147693.0 | 338.795457 |
| 1 | chr1 | chr3 | 314220340 | 127220.0 | 309.070464 |
| 2 | chr1 | chr4 | 306122580 | 123733.0 | 290.960206 |
| 3 | chr1 | chr5 | 299987240 | 112388.0 | 264.008169 |
| 4 | chr1 | chr6 | 294360620 | 119053.0 | 280.474475 |

stats = {}

for cond in conditions:

n\_cis = int(cis\_exp[cond]['count.sum'].sum())

n\_trs = int(trs\_exp[cond]['count.sum'].sum())

stats[long\_names[cond]] = {

'cis': n\_cis,

'trans': n\_trs,

'total': n\_cis + n\_trs,

'cis:trans': n\_cis / n\_trs,

'cis:total': n\_cis / (n\_cis + n\_trs)

}

pd.DataFrame.from\_dict(stats, orient='index')

|  | cis | trans | total | cis:trans | cis:total |
| --- | --- | --- | --- | --- | --- |
| NipblKO | 60644854 | 29374071 | 90018925 | 2.064571 | 0.673690 |
| TAM | 46889387 | 19725414 | 66614801 | 2.377105 | 0.703888 |
| Wildtype | 37717628 | 14296684 | 52014312 | 2.638208 | 0.725139 |

sums = {}

n\_valid = {}

scalings = {}

for cond in conditions:

grouped = cis\_exp[cond].groupby('diag')

n\_valid[cond] = grouped['n\_valid'].sum().values

sums[cond] = grouped['balanced.sum'].sum().values

scalings[cond] = (sums[cond] / n\_valid[cond])

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:8: RuntimeWarning: invalid value encountered in true\_divide

from cooltools.lib import numutils

​

def coarsen\_geometric(sums, counts, n\_bins=100):

"""Re-bin the expected sums into logarithmically growing bins.

"""

dbins = numutils.logbins(1, len(sums), N=n\_bins)

spans = list(zip(dbins[:-1], dbins[1:]))

s = np.array([np.nansum(sums[lo:hi]) for lo, hi in spans])

n = np.array([np.nansum(counts[lo:hi]) for lo, hi in spans])

return dbins, s / n

gs = plt.GridSpec(nrows=2, ncols=1, height\_ratios=[10, 2])

plt.figure(figsize=(8, 15))

​

SMOOTH = False

​

ax1 = plt.subplot(gs[0])

for i, cond in enumerate(conditions):

ref\_point = 200000 // binsize

norm = 1 #scalings[cond][ref\_point]

# cis P(s)

x = np.arange(0, len(scalings[cond]) \* binsize, binsize)

y = scalings[cond] / norm

if SMOOTH:

x, y = coarsen\_geometric(sums[cond], n\_valid[cond], 100)

x \*= binsize

x = x[:-1]

plt.plot(x, y,

color=colors[cond],

label=long\_names[cond])

# average trans levels

for \_, row in trs\_exp[cond].iterrows():

plt.axhline(

(row['balanced.sum']/row['n\_valid']) / norm,

xmin=i/len(conditions),

xmax=(i+1)/len(conditions),

c=colors[cond],

alpha=0.25)

​

plt.xscale('log')

plt.yscale('log')

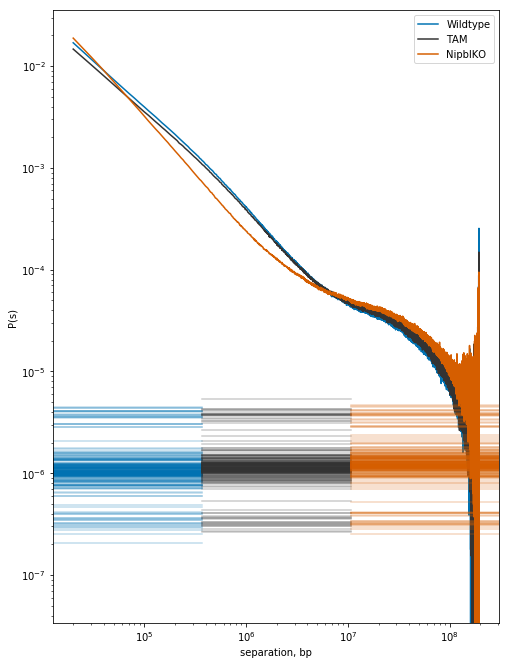
plt.ylabel('P(s)')

plt.xlabel('separation, bp')

plt.legend()

plt.gca().set\_aspect(1)

xlim = plt.xlim()



# Compartmentalization via eigendecomposition

%matplotlib inline

from matplotlib.gridspec import GridSpec

import matplotlib.pyplot as plt

import matplotlib as mpl

import seaborn as sns

mpl.style.use('seaborn-white')

​

import multiprocess as mp

import numpy as np

import pandas as pd

import bioframe

import cooltools

import cooler

mm9 = bioframe.fetch\_chromsizes('mm9')

chromsizes = bioframe.fetch\_chromsizes('mm9')

chromosomes = list(chromsizes.index)

conditions = ['WT', 'T', 'dN']

binsize = 100000

​

cooler\_paths = {

'WT' : f'data/UNTR.{binsize}.cool',

'T' : f'data/TAM.{binsize}.cool',

'dN' : f'data/NIPBL.{binsize}.cool',

}

long\_names = {

'WT': 'Wildtype',

'T' : 'TAM',

'dN': 'NipblKO',

}

pal = sns.color\_palette('colorblind')

colors = {

'WT': pal[0],

'T' : '#333333',

'dN': pal[2],

}

​

clrs = {

cond: cooler.Cooler(cooler\_paths[cond]) for cond in conditions

}

## Get binned GC-content (proxy for the "A" phase)

bins = cooler.binnify(mm9, binsize)

fasta\_records = bioframe.load\_fasta('data/mm9.fa')

bins['GC'] = bioframe.tools.frac\_gc(bins, fasta\_records)

bins.head()

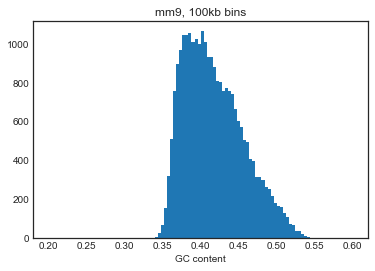
|  | chrom | start | end | GC |
| --- | --- | --- | --- | --- |
| 0 | chr1 | 0 | 100000 | NaN |
| 1 | chr1 | 100000 | 200000 | NaN |
| 2 | chr1 | 200000 | 300000 | NaN |
| 3 | chr1 | 300000 | 400000 | NaN |
| 4 | chr1 | 400000 | 500000 | NaN |

\_=plt.hist(bins['GC'].dropna(), range=(0.2, 0.6), bins=100)

plt.xlabel('GC content')

plt.title(f'mm9, {binsize//1000}kb bins')

<matplotlib.text.Text at 0x7f49199284e0>



## Compute eigenvectors

from cooltools.eigdecomp import cooler\_cis\_eig

​

lam = {}

eigs = {}

​

for cond in conditions:

lam[cond], eigs[cond] = cooler\_cis\_eig(

clrs[cond],

bins,

n\_eigs=3,

phasing\_track\_col='GC',

sort\_metric='var\_explained')

# Save text files

lam[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.eigs.cis.lam.txt', sep='\t')

eigs[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.eigs.cis.vecs.txt', sep='\t', index=False)

# Save bigwig track

bioframe.to\_bigwig(eigs[cond], mm9, f'data/{long\_names[cond]}.{binsize//1000}kb.eigs.cis.vecs.E1.bw', 'E1')

bedGraphToBigWig /tmp/tmp\_k979fat.bg /tmp/tmpdsfw26i1.chrom.sizes data/Wildtype.100kb.eigs.cis.vecs.E1.bw

bedGraphToBigWig /tmp/tmpvygdsjvy.bg /tmp/tmprf3fi3gj.chrom.sizes data/TAM.100kb.eigs.cis.vecs.E1.bw

bedGraphToBigWig /tmp/tmpibx6m9g8.bg /tmp/tmpblzcmzh9.chrom.sizes data/NipblKO.100kb.eigs.cis.vecs.E1.bw

from scipy.stats import rankdata

​

gs = plt.GridSpec(nrows=1, ncols=2)

plt.figure(figsize=(16, 6))

condx, condy = 'WT', 'dN'

​

plt.subplot(gs[0])

lo, hi = -2 , 2

plt.hexbin(

eigs[condx]['E1'],

eigs[condy]['E1'],

vmax=50,

)

plt.xlabel('E1 ' + long\_names[condx])

plt.ylabel('E1 ' + long\_names[condy])

plt.gca().set\_aspect(1)

plt.xlim(lo, hi)

plt.ylim(lo, hi)

plt.axvline(0, c='b', lw=0.5, ls='--')

plt.axhline(0, c='b', lw=0.5, ls='--')

plt.plot([lo, hi], [lo, hi], c='b', lw=0.5, ls='--')

plt.colorbar(shrink=0.6)

​

​

plt.subplot(gs[1])

mask = eigs[condx]['E1'].notnull() & eigs[condy]['E1'].notnull()

vx = eigs[condx]['E1'].loc[mask].values

vy = eigs[condy]['E1'].loc[mask].values

lo, hi = 0 , len(vx)

​

plt.hexbin(

rankdata(vx),

rankdata(vy),

vmax=20,

)

plt.xlabel('E1 rank ' + long\_names[condx])

plt.ylabel('E1 rank ' + long\_names[condy])

plt.gca().set\_aspect(1)

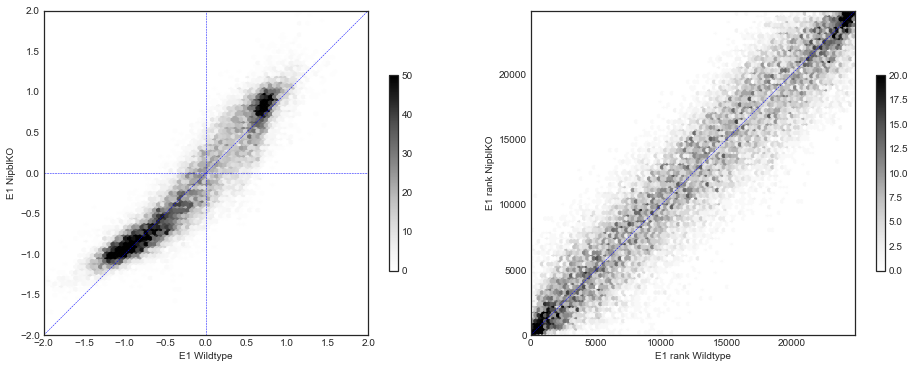
plt.xlim(lo, hi)

plt.ylim(lo, hi)

plt.plot([lo, hi], [lo, hi], c='b', lw=0.5, ls='--')

plt.colorbar(shrink=0.6)

<matplotlib.colorbar.Colorbar at 0x7f4915fcccf8>



## The "saddle" plot

from cooltools import saddle

​

QUANTILE\_BINNING = True

​

binedges = {}

digitized = {}

hist = {}

sums = {}

counts = {}

saddledata = {}

​

gs = plt.GridSpec(nrows=1, ncols=2)

fig = plt.figure(figsize=(12, 6))

histbins = 30

​

for i, cond in enumerate(['WT', 'dN']):

exp = pd.read\_table(f'data/{long\_names[cond]}.{binsize//1000}kb.expected.cis.tsv')

eig = pd.read\_table(f'data/{long\_names[cond]}.{binsize//1000}kb.eigs.cis.vecs.txt')

​

# Determine how to bin the range of the E1 signal

if QUANTILE\_BINNING:

q\_binedges = np.linspace(0, 1, histbins)

binedges[cond] = saddle.quantile(eig['E1'], q\_binedges)

else:

qlo, qhi = saddle.quantile(eig['E1'], [0.02, 0.98]) # trim outliers

binedges[cond] = np.linspace(qlo, qhi, histbins)

​

# Digitize the signal into integers

digitized[cond], hist[cond] = saddle.digitize\_track(

binedges[cond],

track=(eig, 'E1'))

# Construct a function that fetches and calculates observed/expected

getmatrix = saddle.make\_cis\_obsexp\_fetcher(clrs[cond], (exp, 'balanced.avg'))

# Build the saddle histogram

sums[cond], counts[cond] = saddle.make\_saddle(

getmatrix,

binedges[cond],

(digitized[cond], 'E1.d'),

contact\_type='cis')

saddledata[cond] = sums[cond] / counts[cond]

# Make the saddle plot

g = saddle.saddleplot(

q\_binedges if QUANTILE\_BINNING else binedges[cond],

hist[cond],

np.log10(saddledata[cond]),

color=colors[cond],

heatmap\_kws={'vmin': -1, 'vmax': 1},

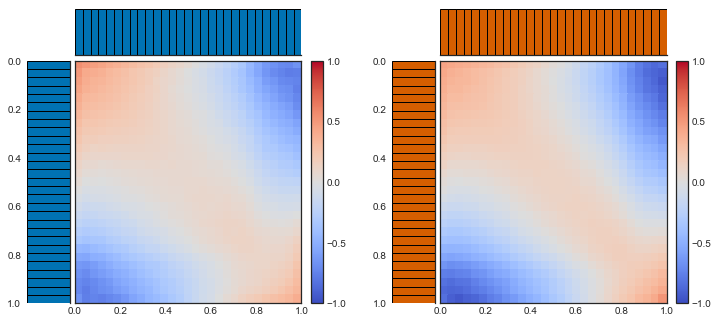
fig=fig, subplot\_spec=gs[i])

/net/proteome/home/nezar/local/devel/cooltools/cooltools/saddle.py:109: RuntimeWarning: invalid value encountered in true\_divide

toeplitz(expected[chrom])

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:42: RuntimeWarning: invalid value encountered in true\_divide

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:48: RuntimeWarning: divide by zero encountered in log10



strength = {

cond: saddle.saddle\_strength(sums[cond], counts[cond])

for cond in ['WT', 'dN']

}

​

gs = plt.GridSpec(nrows=1, ncols=2)

plt.figure(figsize=(14, 6))

​

plt.subplot(gs[0])

x = np.arange(histbins + 2)

for cond in ['WT', 'dN']:

plt.step(x[:-1], strength[cond], where='pre', color=colors[cond], label=long\_names[cond])

​

plt.legend()

plt.xlabel('extent')

plt.ylabel('(AA + BB) / (AB + BA)')

plt.title('saddle strength profile')

plt.axhline(0, c='grey', ls='--', lw=1)

plt.xlim(0, len(x)//2)

​

plt.subplot(gs[1])

plt.step(x[:-1], strength['dN'] / strength['WT'], where='pre', c='k')

plt.axhline(1, c='grey', ls='--', lw=1)

plt.xlim(0, len(x)//2)

plt.xlabel('extent')

plt.ylabel('enrichment')

plt.title('NipblKO / Wildtype')

/net/proteome/home/nezar/local/devel/cooltools/cooltools/saddle.py:443: RuntimeWarning: invalid value encountered in double\_scalars

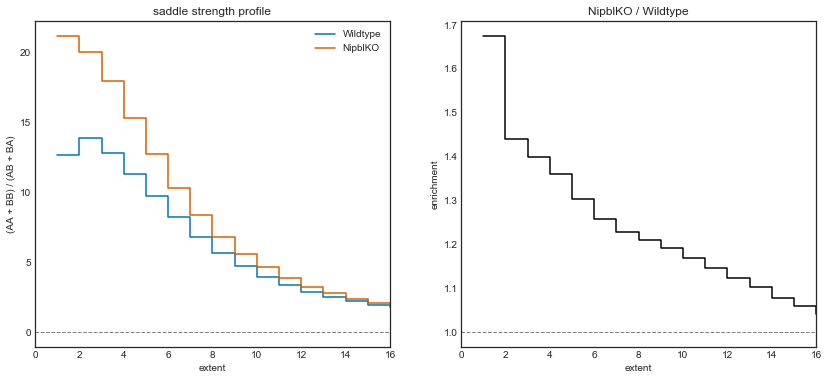
intra = intra\_sum / intra\_count

/net/proteome/home/nezar/local/devel/cooltools/cooltools/saddle.py:447: RuntimeWarning: invalid value encountered in double\_scalars

inter = inter\_sum / inter\_count

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:22: RuntimeWarning: invalid value encountered in true\_divide

<matplotlib.text.Text at 0x7f490f606630>



# Contact-insulating loci

%matplotlib inline

from matplotlib.gridspec import GridSpec

import matplotlib.pyplot as plt

import matplotlib as mpl

import seaborn as sns

mpl.style.use('seaborn-white')

import multiprocess as mp

import numpy as np

import pandas as pd

import bioframe

import cooltools

import cooler

import bbi

mm9 = bioframe.fetch\_chromsizes('mm9')

chromsizes = bioframe.fetch\_chromsizes('mm9')

chromosomes = list(chromsizes.index)

conditions = ['WT', 'T', 'dN']

binsize = 100000

​

cooler\_paths = {

'WT' : f'data/UNTR.{binsize}.cool',

'T' : f'data/TAM.{binsize}.cool',

'dN' : f'data/NIPBL.{binsize}.cool',

}

long\_names = {

'WT': 'Wildtype',

'T' : 'TAM',

'dN': 'NipblKO',

}

pal = sns.color\_palette('colorblind')

colors = {

'WT': pal[0],

'T' : '#333333',

'dN': pal[2],

}

​

clrs = {

cond: cooler.Cooler(cooler\_paths[cond]) for cond in conditions

}

from cooltools.insulation import find\_insulating\_boundaries

from cooltools.directionality import directionality

​

window\_bp = binsize \* 5

​

insul = {}

direc = {}

​

for cond in conditions:

# Diamond insulation score

insul[cond] = find\_insulating\_boundaries(

clrs[cond],

balance='weight',

window\_bp=window\_bp,

min\_dist\_bad\_bin=2,

)

insul[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.insul\_{window\_bp}.tsv', sep='\t')

bioframe.to\_bigwig(insul[cond], mm9,

f'data/{long\_names[cond]}.{binsize//1000}kb.insul\_score\_{window\_bp}.bw',

f'log2\_insulation\_score\_{window\_bp}')

bioframe.to\_bigwig(insul[cond], mm9,

f'data/{long\_names[cond]}.{binsize//1000}kb.insul\_pp\_{window\_bp}.bw',

f'boundary\_strength\_{window\_bp}')

# Directionality Index

direc[cond] = directionality(

clrs[cond],

window\_bp=window\_bp,

min\_dist\_bad\_bin=2,

)

direc[cond].to\_csv(f'data/{long\_names[cond]}.{binsize//1000}kb.direc\_{window\_bp}.tsv', sep='\t')

bioframe.to\_bigwig(direc[cond], mm9,

f'data/{long\_names[cond]}.{binsize//1000}kb.direc\_index\_{window\_bp}.bw',

f'directionality\_index\_{window\_bp}')

bioframe.to\_bigwig(direc[cond], mm9,

f'data/{long\_names[cond]}.{binsize//1000}kb.direc\_ratio\_{window\_bp}.bw',

f'directionality\_ratio\_{window\_bp}')

bedGraphToBigWig /tmp/tmp42su03hm.bg /tmp/tmp92cvxkk5.chrom.sizes data/Wildtype.100kb.insul\_score\_500000.bw

bedGraphToBigWig /tmp/tmp\_31j5aq1.bg /tmp/tmp9hwerl3g.chrom.sizes data/Wildtype.100kb.insul\_pp\_500000.bw

bedGraphToBigWig /tmp/tmpn9r\_qco3.bg /tmp/tmph4y0vmwe.chrom.sizes data/Wildtype.100kb.direc\_index\_500000.bw

bedGraphToBigWig /tmp/tmp82lu15mp.bg /tmp/tmph8h7q9d0.chrom.sizes data/Wildtype.100kb.direc\_ratio\_500000.bw

bedGraphToBigWig /tmp/tmp5smdg3du.bg /tmp/tmp2dohdna7.chrom.sizes data/TAM.100kb.insul\_score\_500000.bw

bedGraphToBigWig /tmp/tmp5wrrnnhk.bg /tmp/tmpcl4me291.chrom.sizes data/TAM.100kb.insul\_pp\_500000.bw

bedGraphToBigWig /tmp/tmpfwt33lb5.bg /tmp/tmplwlvyrpy.chrom.sizes data/TAM.100kb.direc\_index\_500000.bw

bedGraphToBigWig /tmp/tmppgw6jm38.bg /tmp/tmpganyga0r.chrom.sizes data/TAM.100kb.direc\_ratio\_500000.bw

bedGraphToBigWig /tmp/tmpdq9d\_neu.bg /tmp/tmp5ztr97ks.chrom.sizes data/NipblKO.100kb.insul\_score\_500000.bw

bedGraphToBigWig /tmp/tmpebp9f5or.bg /tmp/tmpgz97vez6.chrom.sizes data/NipblKO.100kb.insul\_pp\_500000.bw

bedGraphToBigWig /tmp/tmpgsq9xqy4.bg /tmp/tmpg4k8w6ww.chrom.sizes data/NipblKO.100kb.direc\_index\_500000.bw

bedGraphToBigWig /tmp/tmp5gbxsokb.bg /tmp/tmpf00sut9s.chrom.sizes data/NipblKO.100kb.direc\_ratio\_500000.bw

gs = plt.GridSpec(nrows=1, ncols=2)

plt.figure(figsize=(8\*2, 8))

​

binedges = np.linspace(-2.5, 2.5, 60)

plt.subplot(gs[0])

for cond in conditions:

x = insul[cond][f'log2\_insulation\_score\_{window\_bp}'].values

plt.hist(x[~np.isnan(x)],

bins=binedges,

histtype='step',

lw=2,

label=long\_names[cond],

color=colors[cond])

plt.axvline(0, c='k', ls='--')

plt.legend()

plt.title(f'log2(insul) @{binsize//1000}kb ; window={window\_bp//1000}kb')

​

plt.subplot(gs[1])

binedges = np.linspace(-3, 1, 45)

for cond in conditions[::-1]:

x = insul[cond][f'boundary\_strength\_{window\_bp}'].apply(np.log10).values

plt.hist(x[~np.isnan(x)],

bins=binedges,

histtype='step',

lw=2,

label=long\_names[cond],

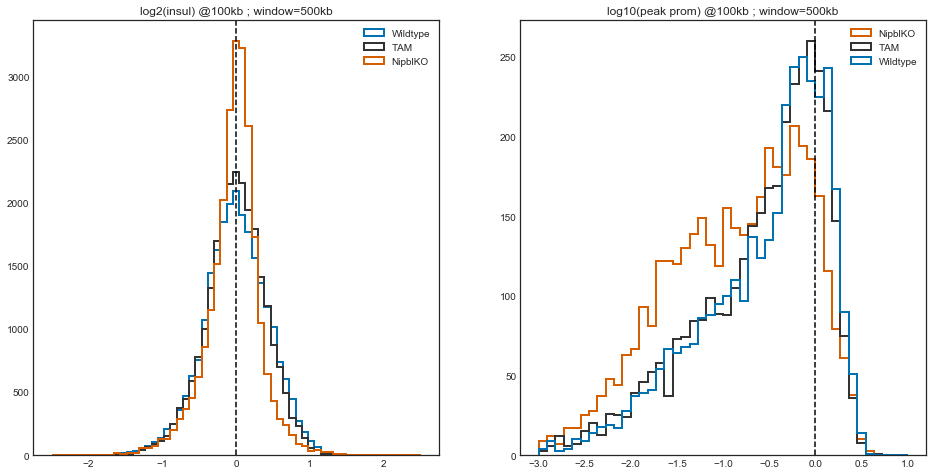
color=colors[cond])

plt.axvline(0, c='k', ls='--')

plt.legend()

plt.title(f'log10(peak prom) @{binsize//1000}kb ; window={window\_bp//1000}kb')

<matplotlib.text.Text at 0x7fa89cb1b9b0>



gs = plt.GridSpec(nrows=2, ncols=2)

plt.figure(figsize=(8\*2, 8\*2))

​

for i, cond in enumerate(conditions):

plt.subplot(gs[i])

plt.scatter(

insul[cond][f'log2\_insulation\_score\_{window\_bp}'].values,

np.log10(insul[cond][f'boundary\_strength\_{window\_bp}']).values,

s=4,

edgecolor='k',

facecolor='none',

alpha=0.1

)

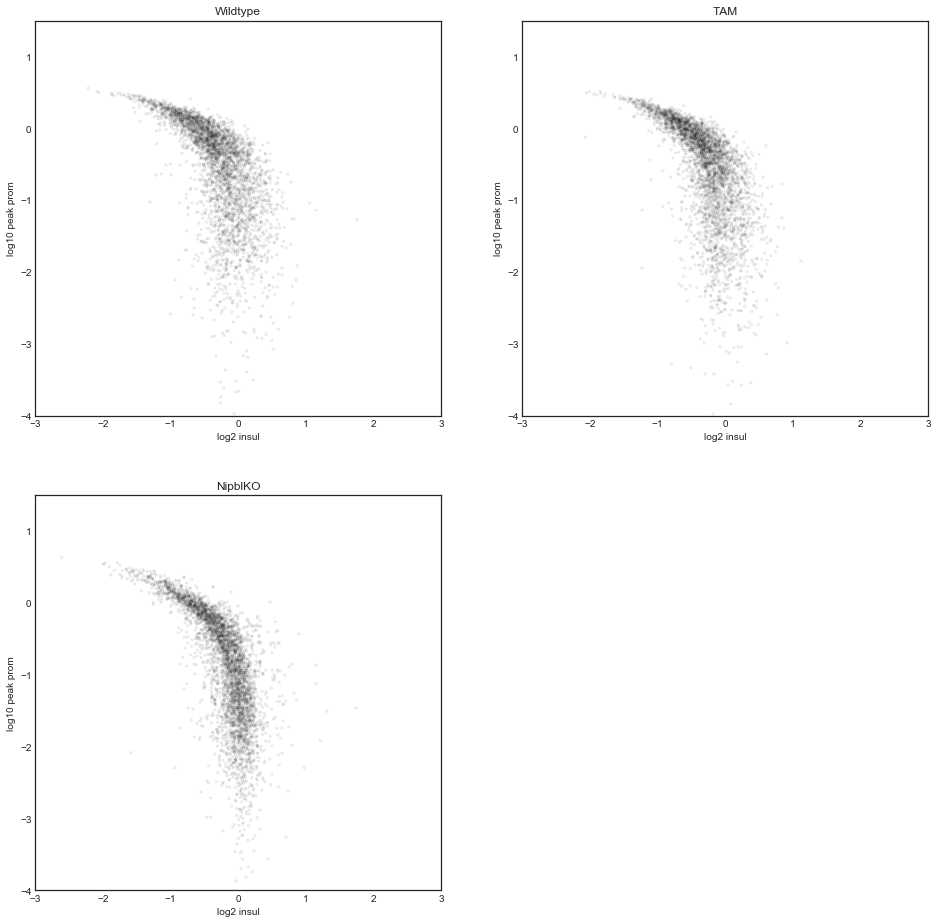
plt.title(long\_names[cond])

plt.xlabel('log2 insul')

plt.ylabel('log10 peak prom')

plt.xlim(-3, 3)

plt.ylim(-4, 1.5)



gs = GridSpec(nrows=2, ncols=2)

​

for i, (condx, condy) in enumerate([('WT', 'T'), ('WT', 'dN')]):

plt.figure(figsize=(2 \* 6, 2 \* 6))

​

score = f'log2\_insulation\_score\_{window\_bp}'

plt.subplot(gs[i, 0])

plt.hexbin(

insul[condx][score],

insul[condy][score],

extent=(-2, 2, -2, 2),

vmin=0,

vmax=150,

cmap=sns.blend\_palette(['w', 'orange', 'r', 'k'], as\_cmap=True),

alpha=0.6,

rasterized=True

)

plt.plot([-2, 2], [-2, 2], c='b', ls='--', lw=0.5)

plt.axvline(0, c='b', lw=0.5, ls='--')

plt.axhline(0, c='b', lw=0.5, ls='--')

plt.gca().set\_aspect(1)

plt.colorbar(shrink=0.8)

plt.xlabel(long\_names[condx])

plt.ylabel(long\_names[condy])

plt.title(score)

​

​

score = f'directionality\_index\_{window\_bp}'

plt.subplot(gs[i, 1])

plt.hexbin(

direc[condx][score],

direc[condy][score],

extent=(-0.25, 0.25, -0.25, 0.25),

vmin=0,

vmax=15,

cmap=sns.blend\_palette(['w', 'orange', 'r', 'k'], as\_cmap=True),

alpha=0.6,

rasterized=True

)

plt.plot([-0.25, 0.25], [-0.25, 0.25], c='b', ls='--', lw=0.5)

plt.axvline(0, c='b', lw=0.5, ls='--')

plt.axhline(0, c='b', lw=0.5, ls='--')

plt.gca().set\_aspect(1)

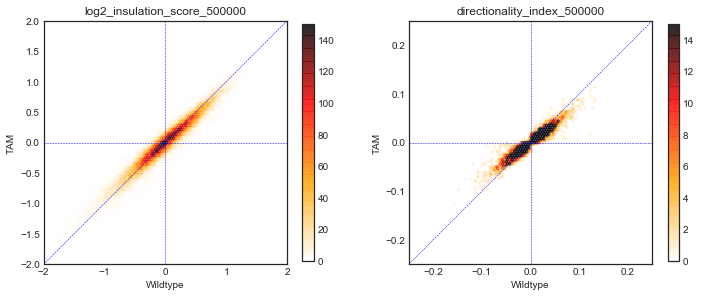
plt.colorbar(shrink=0.8)

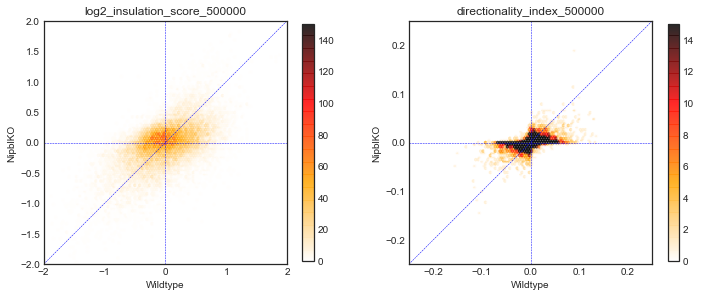
plt.xlabel(long\_names[condx])

plt.ylabel(long\_names[condy])

plt.title(score)

​





peaks = {}

pscore = f'boundary\_strength\_{window\_bp}'

cols = ['chrom', 'start', 'end', pscore]

for cond in conditions:

peaks[cond] = (

insul[cond].dropna(subset=[pscore])[cols]

.sort\_values(pscore, ascending=False) # peaks!

)

wt\_peaks = peaks['WT']

len(peaks['WT']), len(peaks['T']), len(peaks['dN'])

(3457, 3537, 3968)

stacks = {}

nbins = 2000

for i, cond in enumerate(conditions):

mids = (wt\_peaks['start'] + wt\_peaks['end']) // 2

flank = 1000000

bwfile = f'data/{long\_names[cond]}.100kb.insul\_score\_500000.bw'

#bwfile = f'data/{long\_names[cond]}.100kb.direc\_ratio\_500000.bw'

#bwfile = f'data/{long\_names[cond]}.100kb.eigs.cis.vecs.E1.bw'

stacks[cond] = bbi.stackup(bwfile, wt\_peaks['chrom'], mids - flank, mids + flank, bins=nbins)

​

gs = GridSpec(nrows=3, ncols=len(conditions),

height\_ratios=[15, 2, 0.5],

hspace=0)

​

plt.figure(figsize=(3\*len(conditions), 10))

​

X = stacks['WT']

idx = np.argsort(X[:, X.shape[1]//2])

x = np.linspace(-flank/1e6, flank/1e6, nbins)

cmap = plt.cm.get\_cmap('coolwarm')

cmap.set\_bad('#777777')

im\_opts = dict(

vmin=-0.5,

vmax=0.5,

extent=[-flank/1e6, flank/1e6, len(wt\_peaks), 0],

cmap=cmap

)

​

for i, name in enumerate(stacks):

# heatmap

ax = ax1 = plt.subplot(gs[0, i])

X = stacks[name]

img = ax.matshow(X[idx, :], \*\*im\_opts, rasterized=True)

ax.axvline(0, c='grey', lw=0.5)

ax.grid('off')

ax.set\_aspect('auto')

ax.set\_title(long\_names[name])

if i > 0:

ax.yaxis.set\_visible(False)

# summary

ax = plt.subplot(gs[1, i], sharex=ax1)

ax.axhline(0, c='#777777', lw=1, ls='--')

ax.plot(x, np.nanmean(stacks[name], axis=0), c='k', lw=2)

ax.set\_xlim(-flank/1e6, flank/1e6)

ax.xaxis.set\_visible(False)

ax.set\_ylim(-0.5, 0.5)

if i > 0:

ax.yaxis.set\_visible(False)

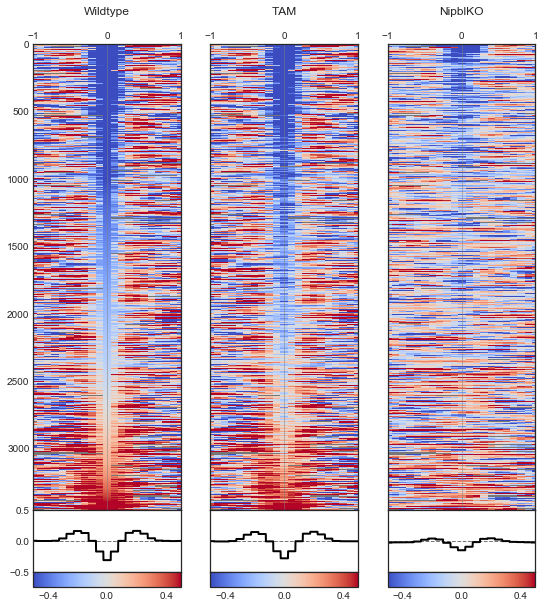
# color bar

cax = plt.subplot(gs[2, i])

cb = plt.colorbar(img, cax=cax, orientation='horizontal')

cb.locator = mpl.ticker.MaxNLocator(nbins=3)

cb.update\_ticks()



stacks = {}

nbins = 2000

for i, cond in enumerate(conditions):

mids = (wt\_peaks['start'] + wt\_peaks['end']) // 2

flank = 1000000

bwfile = f'data/{long\_names[cond]}.100kb.direc\_ratio\_500000.bw'

stacks[cond] = bbi.stackup(bwfile, wt\_peaks['chrom'], mids - flank, mids + flank, bins=nbins)

​

gs = GridSpec(nrows=3, ncols=len(conditions),

height\_ratios=[15, 2, 0.5],

hspace=0)

​

plt.figure(figsize=(3\*len(conditions), 10))

​

X = stacks['WT']

idx = np.argsort(X[:, X.shape[1]//2])

x = np.linspace(-flank/1e6, flank/1e6, nbins)

cmap = plt.cm.get\_cmap('coolwarm')

cmap.set\_bad('#777777')

im\_opts = dict(

vmin=-0.5,

vmax=0.5,

extent=[-flank/1e6, flank/1e6, len(wt\_peaks), 0],

cmap=cmap

)

​

for i, name in enumerate(stacks):

# heatmap

ax = ax1 = plt.subplot(gs[0, i])

X = stacks[name]

img = ax.matshow(X[idx, :], \*\*im\_opts, rasterized=True)

ax.axvline(0, c='grey', lw=0.5)

ax.grid('off')

ax.set\_aspect('auto')

ax.set\_title(long\_names[name])

if i > 0:

ax.yaxis.set\_visible(False)

# summary

ax = plt.subplot(gs[1, i], sharex=ax1)

ax.axhline(0, c='#777777', lw=1, ls='--')

ax.plot(x, np.nanmean(stacks[name], axis=0), c='k', lw=2)

ax.set\_xlim(-flank/1e6, flank/1e6)

ax.xaxis.set\_visible(False)

ax.set\_ylim(-0.5, 0.5)

if i > 0:

ax.yaxis.set\_visible(False)

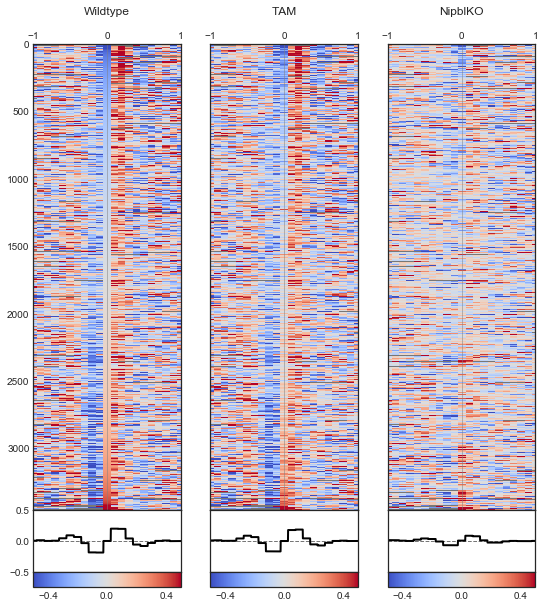
# color bar

cax = plt.subplot(gs[2, i])

cb = plt.colorbar(img, cax=cax, orientation='horizontal')

cb.locator = mpl.ticker.MaxNLocator(nbins=3)

cb.update\_ticks()



# Behold the Mighty Pileup

%matplotlib inline

from matplotlib.gridspec import GridSpec

import matplotlib.pyplot as plt

import matplotlib as mpl

import seaborn as sns

mpl.style.use('seaborn-white')

import multiprocess as mp

import numpy as np

import pandas as pd

import bioframe

import cooltools

import cooler

import bbi

mm9 = bioframe.fetch\_chromsizes('mm9')

chromsizes = bioframe.fetch\_chromsizes('mm9')

chromosomes = list(chromsizes.index)

conditions = ['WT', 'dN']

binsize = 10000

​

cooler\_paths = {

'WT' : f'data/UNTR.{binsize}.cool',

'T' : f'data/TAM.{binsize}.cool',

'dN' : f'data/NIPBL.{binsize}.cool',

}

long\_names = {

'WT': 'Wildtype',

'T' : 'TAM',

'dN': 'NipblKO',

}

pal = sns.color\_palette('colorblind')

colors = {

'WT': pal[0],

'T' : '#333333',

'dN': pal[2],

}

​

clrs = {

cond: cooler.Cooler(cooler\_paths[cond]) for cond in conditions

}

## Single landmark pileup

from cooltools import snipping

ctcf = pd.read\_table('data/CtcfCtrl.mm9\_\_VS\_\_InputCtrl.mm9.narrowPeak\_with\_motif.txt.gz')

ctcf.head()

|  | chrom | start | end | name | score | strand | fc | -log10p | -log10q | relSummit | chrom\_m | start\_m | end\_m | name\_m | score\_m | strand\_m | pval\_m |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | chr14 | 73909345 | 73910139 | Peak\_1 | 2864 | . | 66.97627 | 286.44055 | 277.00751 | 386 | chr14 | 73909703 | 73909716 | CTCF\_mouse | 12.8061 | - | 1.610000e-05 |
| 1 | chr8 | 73292691 | 73293764 | Peak\_10 | 2288 | . | 59.05101 | 228.89639 | 222.17598 | 738 | chr8 | 73293454 | 73293467 | CTCF\_mouse | 14.2242 | + | 6.080000e-06 |
| 2 | chr10 | 99078331 | 99079198 | Peak\_100 | 1567 | . | 40.02444 | 156.79012 | 151.30066 | 303 | chr10 | 99078650 | 99078663 | CTCF\_mouse | 19.7758 | + | 6.020000e-08 |
| 3 | chr19 | 44437901 | 44438397 | Peak\_1000 | 741 | . | 20.84388 | 74.12866 | 69.69677 | 273 | chr19 | 44438190 | 44438203 | CTCF\_mouse | 15.1212 | - | 3.280000e-06 |
| 4 | chr1 | 145853760 | 145854152 | Peak\_10000 | 355 | . | 12.38549 | 35.59266 | 32.22501 | 189 | chr1 | 145853924 | 145853937 | CTCF\_mouse | 17.6242 | + | 3.710000e-07 |

sites = ctcf.sort\_values('fc', ascending=False).iloc[:1000]

sites.head()

|  | chrom | start | end | name | score | strand | fc | -log10p | -log10q | relSummit | chrom\_m | start\_m | end\_m | name\_m | score\_m | strand\_m | pval\_m |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39612 | chr2 | 28440591 | 28441124 | Peak\_7 | 2444 | . | 67.20400 | 244.48576 | 237.52400 | 289 | chr2 | 28440889 | 28440902 | CTCF\_mouse | 17.1879 | + | 5.530000e-07 |
| 0 | chr14 | 73909345 | 73910139 | Peak\_1 | 2864 | . | 66.97627 | 286.44055 | 277.00751 | 386 | chr14 | 73909703 | 73909716 | CTCF\_mouse | 12.8061 | - | 1.610000e-05 |
| 8403 | chr12 | 85930537 | 85931240 | Peak\_18 | 2221 | . | 66.84265 | 222.16884 | 215.60326 | 320 | chr12 | 85930837 | 85930850 | CTCF\_mouse | 17.9212 | + | 2.800000e-07 |
| 5278 | chr11 | 78267090 | 78267719 | Peak\_15 | 2240 | . | 66.48611 | 224.05933 | 217.43312 | 301 | chr11 | 78267350 | 78267363 | CTCF\_mouse | 17.9636 | - | 2.770000e-07 |
| 40800 | chr11 | 98203663 | 98204470 | Peak\_8 | 2370 | . | 65.88831 | 237.03339 | 230.20671 | 426 | chr11 | 98204100 | 98204113 | CTCF\_mouse | 20.5697 | - | 2.070000e-08 |

supports = [(chrom, 0, chromsizes[chrom]) for chrom in chromosomes]

​

flank = 600000

windows = snipping.make\_bin\_aligned\_windows(

binsize,

sites['chrom'],

(sites['start\_m'] + sites['end\_m'])//2,

flank\_bp=flank)

windows['strand'] = sites['strand\_m']

windows = snipping.assign\_regions(windows, supports)

windows = windows.dropna()

​

print(len(windows), 'windows, after assigning supports')

windows.head()

1000 windows, after assigning supports

|  | chrom | start | end | lo | hi | strand | region |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39612 | chr2 | 27840000 | 29050000 | 2784 | 2905 | + | chr2:0-181748087 |
| 0 | chr14 | 73300000 | 74510000 | 7330 | 7451 | - | chr14:0-125194864 |
| 8403 | chr12 | 85330000 | 86540000 | 8533 | 8654 | + | chr12:0-121257530 |
| 5278 | chr11 | 77660000 | 78870000 | 7766 | 7887 | - | chr11:0-121843856 |
| 40800 | chr11 | 97600000 | 98810000 | 9760 | 9881 | - | chr11:0-121843856 |

stacks = {}

piles = {}

for cond in conditions:

expected = pd.read\_table(f'data/{long\_names[cond]}.{binsize//1000}kb.expected.cis.tsv')

snipper = snipping.ObsExpSnipper(clrs[cond], expected)

stack = snipping.pileup(windows, snipper.select, snipper.snip)

# mirror reflect snippets whose feature is on the opposite strand

mask = np.array(windows.strand == '+', dtype=bool)

stack[:, :, mask] = stack[::-1, ::-1, mask]

stacks[cond] = stack

piles[cond] = np.nanmean(stack, axis=2)

/home/nezar/miniconda3/envs/py36/lib/python3.6/site-packages/ipykernel\_launcher.py:14: RuntimeWarning: Mean of empty slice

gs = GridSpec(nrows=1, ncols=len(conditions) + 1, width\_ratios=[20] \* len(conditions) + [1])

plt.figure(figsize=(5 \* len(conditions), 5))

​

opts = dict(

vmin=-0.75,

vmax=0.75,

extent=[-flank//1000, flank//1000, -flank//1000, flank//1000],

cmap='coolwarm'

)

​

for i, cond in enumerate(conditions):

ax = plt.subplot(gs[i])

img = ax.matshow(

np.log2(piles[cond]),

\*\*opts)

ax.xaxis.tick\_bottom()

if i > 0:

ax.yaxis.set\_visible(False)

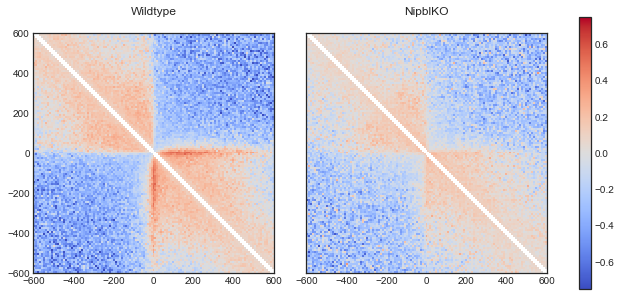
plt.title(long\_names[cond])

​

ax = plt.subplot(gs[len(conditions)])

plt.colorbar(img, cax=ax)

<matplotlib.colorbar.Colorbar at 0x7fa2ad90a128>



## Inspect examples

from ipywidgets import interact

​

gs = GridSpec(nrows=1, ncols=len(conditions) + 1, width\_ratios=[20] \* len(conditions) + [1])

n\_examples = len(windows)

​

@interact(i=(0, n\_examples-1))

def f(i):

plt.figure(figsize=(5 \* len(conditions), 5))

for j, cond in enumerate(conditions):

ax = plt.subplot(gs[j])

img = ax.matshow(

np.log2(stacks[cond][:, :, i]),

\*\*opts)

ax.xaxis.tick\_bottom()

if i > 0:

ax.yaxis.set\_visible(False)

plt.title(long\_names[cond])

plt.axvline(0, c='g', ls='--')

plt.axhline(0, c='g', ls='--')

Failed to display Jupyter Widget of type interactive.

If you're reading this message in the Jupyter Notebook or JupyterLab Notebook, it may mean that the widgets JavaScript is still loading. If this message persists, it likely means that the widgets JavaScript library is either not installed or not enabled. See the [Jupyter Widgets Documentation](https://ipywidgets.readthedocs.io/en/stable/user_install.html) for setup instructions.

If you're reading this message in another frontend (for example, a static rendering on GitHub or [NBViewer](https://nbviewer.jupyter.org/)), it may mean that your frontend doesn't currently support widgets.

## Piling up paired landmarks

anchor\_dist = 300000

anchor\_flank = 10000

sites = pd.read\_table('data/ctcf-sites.paired.300kb\_flank10kb.tsv.1')

​

# "convergent" orientation of paired CTCF motifs

sites = sites[(sites['strand1'] == '+') & (sites['strand2'] == '-')]

​

print(len(sites))

sites.head()

snippet\_flank = 250000

​

windows1 = snipping.make\_bin\_aligned\_windows(

binsize,

sites['chrom1'],

sites['mid1'],

flank\_bp=snippet\_flank)

windows1['strand'] = sites['strand1']

​

windows2 = snipping.make\_bin\_aligned\_windows(

binsize,

sites['chrom2'],

sites['mid2'],

flank\_bp=snippet\_flank)

windows2['strand'] = sites['strand2']

​

windows = pd.merge(windows1, windows2, left\_index=True, right\_index=True, suffixes=('1', '2'))

windows = snipping.assign\_regions(windows, supports)

windows = windows.dropna()

windows.head()

|  | chrom1 | start1 | end1 | lo1 | hi1 | strand1 | chrom2 | start2 | end2 | lo2 | hi2 | strand2 | region |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | chr11 | 97950000 | 98460000 | 9795 | 9846 | + | chr11 | 98360000 | 98870000 | 9836 | 9887 | - | chr11:0-121843856 |
| 1 | chr2 | 27800000 | 28310000 | 2780 | 2831 | + | chr2 | 28190000 | 28700000 | 2819 | 2870 | - | chr2:0-181748087 |
| 2 | chr8 | 72650000 | 73160000 | 7265 | 7316 | + | chr8 | 73040000 | 73550000 | 7304 | 7355 | - | chr8:0-131738871 |
| 3 | chr7 | 74120000 | 74630000 | 7412 | 7463 | + | chr7 | 74520000 | 75030000 | 7452 | 7503 | - | chr7:0-152524553 |
| 4 | chr9 | 77190000 | 77700000 | 7719 | 7770 | + | chr9 | 77600000 | 78110000 | 7760 | 7811 | - | chr9:0-124076172 |

stacks = {}

piles = {}

for cond in conditions:

expected = pd.read\_table(f'data/{long\_names[cond]}.{binsize//1000}kb.expected.cis.tsv')

snipper = snipping.ObsExpSnipper(clrs[cond], expected)

stack = snipping.pileup(windows, snipper.select, snipper.snip)

stacks[cond] = stack

piles[cond] = np.nanmean(stack, axis=2)

gs = plt.GridSpec(nrows=1, ncols=len(conditions) + 1, width\_ratios=[20] \* len(conditions) + [1])

plt.figure(figsize=(6 \* len(conditions), 6))

​

opts = dict(

vmin=-0.5,

vmax=0.5,

extent=[-flank//1000, flank//1000, -flank//1000, flank//1000],

cmap='coolwarm'

)

​

for i, cond in enumerate(conditions):

ax = plt.subplot(gs[i])

img = ax.matshow(

np.log2(np.nanmean(stacks[cond], axis=2)), #piles[cond]),

\*\*opts)

ax.xaxis.tick\_bottom()

if i > 0:

ax.yaxis.set\_visible(False)

plt.title(long\_names[cond])

​

ax = plt.subplot(gs[len(conditions)])

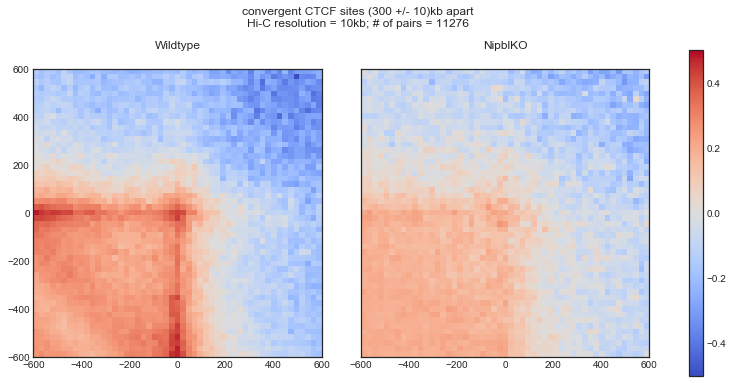
plt.colorbar(img, cax=ax)

​

plt.suptitle(f'convergent CTCF sites ({anchor\_dist//1000} +/- {anchor\_flank//1000})kb apart\n'

f'Hi-C resolution = {binsize//1000}kb; # of pairs = {len(windows)}')

<matplotlib.text.Text at 0x7fa2abd45eb8>



# Mapping Hi-C data with bwa and pairsamtools

import os

​

if os.path.exists('mapping'):

!rm -rf mapping

!mkdir -p mapping

cd mapping

/media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping

### Download sample yeast reads

%%bash

​

curl -LkSs https://api.github.com/repos/mirnylab/distiller-test-data/tarball | tar -zxf -

#mv $(ls -d mirnylab-distiller-test-data-\* | head -n 1)/genome .

#mv $(ls -d mirnylab-distiller-test-data-\* | head -n 1)/fastq .

​

mv /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/open2c-distiller-test-data-0e8b76d/genome .

mv /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/open2c-distiller-test-data-0e8b76d/fastq .

​

ll genome/

итого 24579

-rwxrwxrwx 1 nastyslav 229 июл 19 2018 sacCer3.chrom.sizes\*

-rwxrwxrwx 1 nastyslav 3876091 июл 19 2018 sacCer3.fa.gz\*

-rwxrwxrwx 1 nastyslav 14 июл 19 2018 sacCer3.fa.gz.amb\*

-rwxrwxrwx 1 nastyslav 562 июл 19 2018 sacCer3.fa.gz.ann\*

-rwxrwxrwx 1 nastyslav 12157188 июл 19 2018 sacCer3.fa.gz.bwt\*

-rwxrwxrwx 1 nastyslav 3039278 июл 19 2018 sacCer3.fa.gz.pac\*

-rwxrwxrwx 1 nastyslav 6078608 июл 19 2018 sacCer3.fa.gz.sa\*

-rwxrwxrwx 1 nastyslav 229 июл 19 2018 sacCer3.reduced.chrom.sizes\*

ll fastq/\*/\*

fastq/MATalpha\_R1/lane1:

итого 760

-rwxrwxrwx 1 nastyslav 379902 июл 19 2018 SRR2601842\_1.fastq.gz\*

-rwxrwxrwx 1 nastyslav 395962 июл 19 2018 SRR2601842\_2.fastq.gz\*

fastq/MATalpha\_R1/lane2:

итого 632

-rwxrwxrwx 1 nastyslav 326486 июл 19 2018 SRR2601843\_1.fastq.gz\*

-rwxrwxrwx 1 nastyslav 315563 июл 19 2018 SRR2601843\_2.fastq.gz\*

fastq/MATalpha\_R2/lane1:

итого 824

-rwxrwxrwx 1 nastyslav 410294 июл 19 2018 SRR2601845\_1.fastq.gz\*

-rwxrwxrwx 1 nastyslav 426485 июл 19 2018 SRR2601845\_2.fastq.gz\*

fastq/MATa\_R1/lane1:

итого 684

-rwxrwxrwx 1 nastyslav 344135 июл 19 2018 SRR2601848\_1.fastq.gz\*

-rwxrwxrwx 1 nastyslav 349946 июл 19 2018 SRR2601848\_2.fastq.gz\*

fastq/MATa\_R2/lane1:

итого 848

-rwxrwxrwx 1 nastyslav 424043 июл 19 2018 SRR2601851\_1.fastq.gz\*

-rwxrwxrwx 1 nastyslav 438318 июл 19 2018 SRR2601851\_2.fastq.gz\*

### Simple pipeline for a single run or part of a run

%%bash

​

set -o errexit

set -o nounset

set -o pipefail

​

# INDEX='./genome/sacCer3.fa.gz'

# CHROMSIZES='./genome/sacCer3.chrom.sizes'

# FASTQ1='./fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz'

# FASTQ2='./fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz'

​

​

INDEX='/media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/genome/sacCer3.fa.gz'

CHROMSIZES='/media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/genome/sacCer3.chrom.sizes'

FASTQ1='/media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz'

FASTQ2='/media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz'

OUTPREFIX='MATalpha\_R1'

​

N\_THREADS=8

​

UNMAPPED\_SAM\_PATH=${OUTPREFIX}.unmapped.bam

UNMAPPED\_PAIRS\_PATH=${OUTPREFIX}.unmapped.pairs.gz

NODUPS\_SAM\_PATH=${OUTPREFIX}.nodups.bam

NODUPS\_PAIRS\_PATH=${OUTPREFIX}.nodups.pairs.gz

DUPS\_SAM\_PATH=${OUTPREFIX}.dups.bam

DUPS\_PAIRS\_PATH=${OUTPREFIX}.dups.pairs.gz

​

bwa mem -SPM -t "${N\_THREADS}" "${INDEX}" "${FASTQ1}" "${FASTQ2}" | {

# Classify Hi-C molecules as unmapped/single-sided/multimapped/chimeric/etc

# and output one line per read, containing the following, separated by \\v:

# \* triu-flipped pairs

# \* read id

# \* type of a Hi-C molecule

# \* corresponding sam entries

pairtools parse -c ${CHROMSIZES}

} | {

# Block-sort pairs together with SAM entries

pairtools sort

} | {

# Set unmapped and ambiguous reads aside

pairtools select '(pair\_type == "CX") or (pair\_type == "LL")' \

--output-rest >( pairtools split \

--output-pairs ${UNMAPPED\_PAIRS\_PATH} \

--output-sam ${UNMAPPED\_SAM\_PATH} )

} | {

# Remove duplicates

pairtools dedup \

--output \

>( pairtools split \

--output-pairs ${NODUPS\_PAIRS\_PATH} \

--output-sam ${NODUPS\_SAM\_PATH} ) \

--output-dups \

>( pairtools markasdup \

| pairtools split \

--output-pairs ${DUPS\_PAIRS\_PATH} \

--output-sam ${DUPS\_SAM\_PATH} )

}

[M::bwa\_idx\_load\_from\_disk] read 0 ALT contigs

[M::process] read 20000 sequences (1000000 bp)...

[M::mem\_pestat] # candidate unique pairs for (FF, FR, RF, RR): (0, 0, 0, 0)

[M::mem\_pestat] skip orientation FF as there are not enough pairs

[M::mem\_pestat] skip orientation FR as there are not enough pairs

[M::mem\_pestat] skip orientation RF as there are not enough pairs

[M::mem\_pestat] skip orientation RR as there are not enough pairs

[M::mem\_process\_seqs] Processed 20000 reads in 0.982 CPU sec, 0.694 real sec

[main] Version: 0.7.18-r1243-dirty

[main] CMD: bwa mem -SPM -t 8 /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/genome/sacCer3.fa.gz /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz

[main] Real time: 1.311 sec; CPU: 1.064 sec

# ll

!ls -lh

итого 561K

drwxrwxrwx 1 nastyslav nastyslav 448 июл 19 2018 fastq

drwxrwxrwx 1 nastyslav nastyslav 520 июл 19 2018 genome

-rwxrwxrwx 1 nastyslav nastyslav 900 июн 9 17:25 MATalpha\_R1.dups.bam

-rwxrwxrwx 1 nastyslav nastyslav 926 июн 9 17:25 MATalpha\_R1.dups.pairs.gz

-rwxrwxrwx 1 nastyslav nastyslav 883 июн 9 17:25 MATalpha\_R1.nodups.bam

-rwxrwxrwx 1 nastyslav nastyslav 909 июн 9 17:25 MATalpha\_R1.nodups.pairs.gz

-rwxrwxrwx 1 nastyslav nastyslav 89 июн 9 17:27 MATalpha\_R1.nodups.pairs.gz.px2

-rwxrwxrwx 1 nastyslav nastyslav 460K июн 9 17:25 MATalpha\_R1.unmapped.bam

-rwxrwxrwx 1 nastyslav nastyslav 84K июн 9 17:25 MATalpha\_R1.unmapped.pairs.gz

drwxrwxrwx 1 nastyslav nastyslav 432 июн 9 17:25 open2c-distiller-test-data-0e8b76d

!samtools view MATalpha\_R1.nodups.bam | head

!samtools view MATalpha\_R1.dups.bam | head

!zcat MATalpha\_R1.nodups.pairs.gz | head -n 50

## pairs format v1.0.0

#sorted: chr1-chr2-pos1-pos2

#shape: upper triangle

#genome\_assembly: unknown

#chromsize: chrIV 1531933

#chromsize: chrXV 1091291

#chromsize: chrVII 1090940

#chromsize: chrXII 1078177

#chromsize: chrXVI 948066

#chromsize: chrXIII 924431

#chromsize: chrII 813184

#chromsize: chrXIV 784333

#chromsize: chrX 745751

#chromsize: chrXI 666816

#chromsize: chrV 576874

#chromsize: chrVIII 562643

#chromsize: chrIX 439888

#chromsize: chrIII 316620

#chromsize: chrVI 270161

#chromsize: chrI 230218

#chromsize: chrM 85779

#samheader: @SQ SN:chrI LN:230218

#samheader: @SQ SN:chrII LN:813184

#samheader: @SQ SN:chrIII LN:316620

#samheader: @SQ SN:chrIV LN:1531933

#samheader: @SQ SN:chrIX LN:439888

#samheader: @SQ SN:chrM LN:85779

#samheader: @SQ SN:chrV LN:576874

#samheader: @SQ SN:chrVI LN:270161

#samheader: @SQ SN:chrVII LN:1090940

#samheader: @SQ SN:chrVIII LN:562643

#samheader: @SQ SN:chrX LN:745751

#samheader: @SQ SN:chrXI LN:666816

#samheader: @SQ SN:chrXII LN:1078177

#samheader: @SQ SN:chrXIII LN:924431

#samheader: @SQ SN:chrXIV LN:784333

#samheader: @SQ SN:chrXV LN:1091291

#samheader: @SQ SN:chrXVI LN:948066

#samheader: @HD VN:1.5 SO:unsorted GO:query

#samheader: @PG ID:bwa PN:bwa VN:0.7.18-r1243-dirty CL:bwa mem -SPM -t 8 /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/genome/sacCer3.fa.gz /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/fastq/MATalpha\_R1/lane1/SRR2601842\_1.fastq.gz

#samheader: @PG ID:pairtools\_parse PN:pairtools\_parse CL:/home/nastyslav/anaconda3/envs/bootcamp/bin/pairtools parse -c /media/nastyslav/F29E58A99E5867DD/Users/nasty/Desktop/Магистратура/bootcamp/hic-data-analysis-bootcamp/notebooks/mapping/genome/sacCer3.chrom.sizes PP:bwa VN:0.3.0

#samheader: @PG ID:pairtools\_sort PN:pairtools\_sort CL:/home/nastyslav/anaconda3/envs/bootcamp/bin/pairtools sort PP:pairtools\_parse VN:0.3.0

#samheader: @PG ID:pairtools\_select PN:pairtools\_select CL:/home/nastyslav/anaconda3/envs/bootcamp/bin/pairtools select (pair\_type == "CX") or (pair\_type == "LL") --output-rest /dev/fd/63 PP:pairtools\_sort VN:0.3.0

#samheader: @PG ID:pairtools\_dedup PN:pairtools\_dedup CL:/home/nastyslav/anaconda3/envs/bootcamp/bin/pairtools dedup --output /dev/fd/63 --output-dups /dev/fd/62 PP:pairtools\_select VN:0.3.0

#samheader: @PG ID:pairtools\_split PN:pairtools\_split CL:/home/nastyslav/anaconda3/envs/bootcamp/bin/pairtools split --output-pairs MATalpha\_R1.nodups.pairs.gz --output-sam MATalpha\_R1.nodups.bam PP:pairtools\_dedup VN:0.3.0

#columns: readID chrom1 pos1 chrom2 pos2 strand1 strand2 pair\_type

### Index the deduped pairs file

!pairix MATalpha\_R1.nodups.pairs.gz

ll

итого 561

drwxrwxrwx 1 nastyslav 448 июл 19 2018 fastq/

drwxrwxrwx 1 nastyslav 520 июл 19 2018 genome/

-rwxrwxrwx 1 nastyslav 900 июн 9 17:25 MATalpha\_R1.dups.bam\*

-rwxrwxrwx 1 nastyslav 926 июн 9 17:25 MATalpha\_R1.dups.pairs.gz\*

-rwxrwxrwx 1 nastyslav 883 июн 9 17:25 MATalpha\_R1.nodups.bam\*

-rwxrwxrwx 1 nastyslav 909 июн 9 17:25 MATalpha\_R1.nodups.pairs.gz\*

-rwxrwxrwx 1 nastyslav 470221 июн 9 17:25 MATalpha\_R1.unmapped.bam\*

-rwxrwxrwx 1 nastyslav 85279 июн 9 17:25 MATalpha\_R1.unmapped.pairs.gz\*

drwxrwxrwx 1 nastyslav 432 июн 9 17:25 open2c-distiller-test-data-0e8b76d/

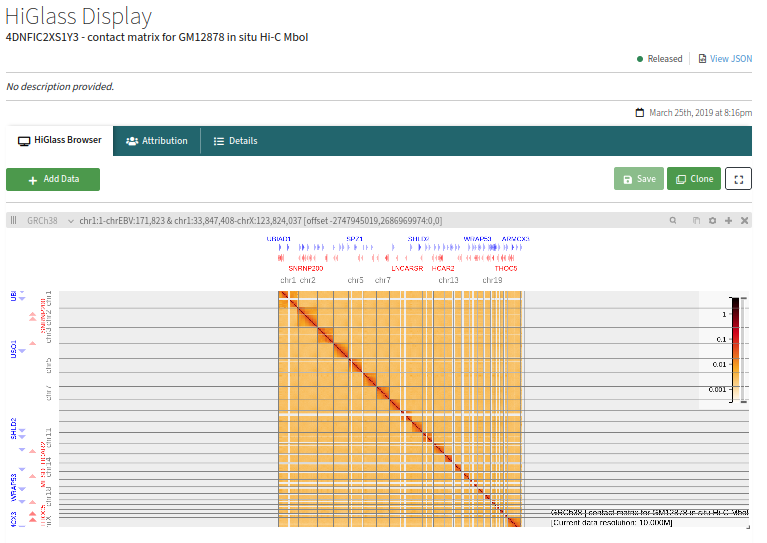
!pairix MATalpha\_R1.nodups.pairs.gz 'chrI|chrI'

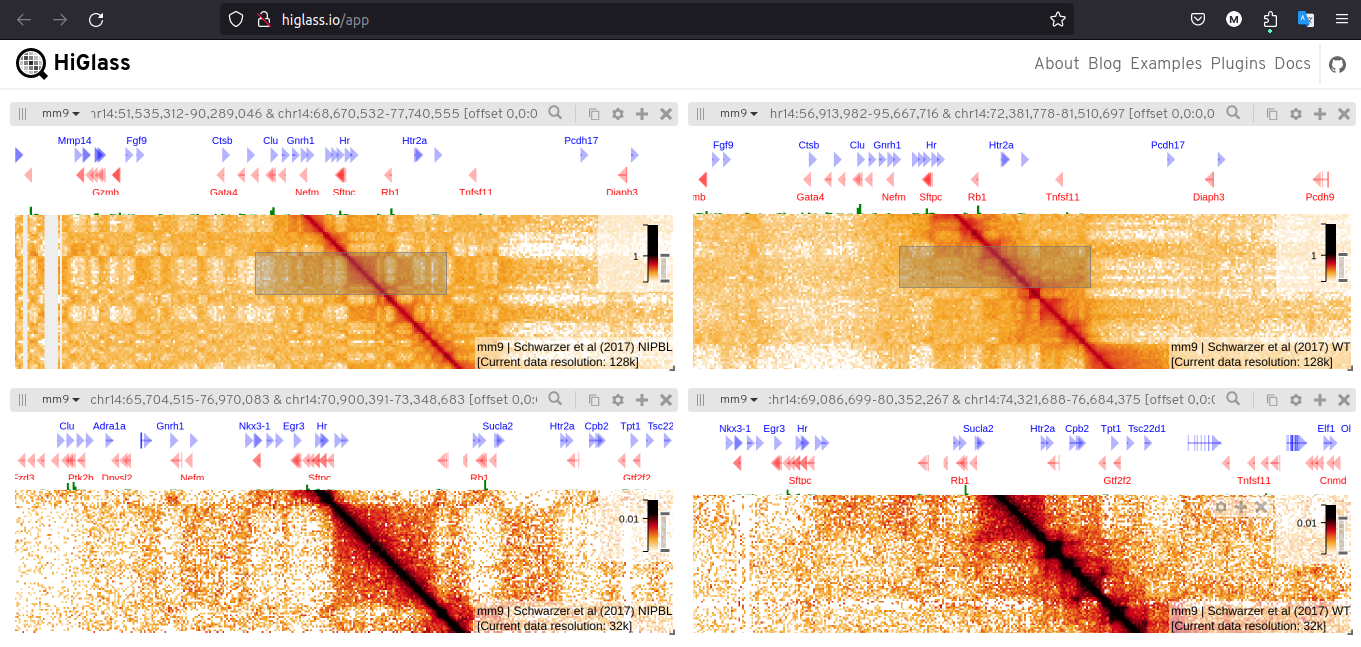
**Визуализация и анализ:**

**Для визуализации контактной матрицы использован инструмент HiGlass.**

**Проведен предварительный анализ полученной матрицы, включая фильтрацию, балансировку и визуализацию.**

**Результаты представлены в виде интерактивных контактных карт, позволяющих исследовать пространственную организацию генома.**

.



http://higlass.io/l/?d=cCq4OQjLSmi8hholmLFssA