# hicstuff Documentation

*Release 3.2.1*

**Lyam Baudry**

**Sep 06, 2023**

# Contents:


**1 Introduction** **3**
- 1.1 Installation . . . 3
- 1.2 Quickstart . . . 3
- 1.3 hicstuff command line interface demo . . . 4
- 1.4 hicstuff API demo . . . 7

**2 Reference API** **13**
- 2.1 hicstuff package . . . 13

**3 Indices and tables** **55**

**Python Module Index** **57**

**Index** **59**

hicstuff is a python library to generate and manipulate Hi-C matrices. It aims to be easy to use and install. You can follow hicstuff's development on Github .

CHAPTER 1

# Introduction

## 1.1 Installation

First, make sure you have the third party dependencies:

- samtools
- minimap2
- bowtie2

You can then install hicstuff latest stable version using pip:

```
pip3 install --user hicstuff
```

Or the latest development (unstable) version using:

```
pip3 install -e git+https://github.com/koszullab/hicstuff.git@master#egg=hicstuff
```

## 1.2 Quickstart

The fastest way to generate Hi-C matrices is to use the *hicstuff pipeline* command:

```
hicstuff pipeline -g bt2_index for.fq rev.fq
```

However, you most likely want to have a look at the command line help to select appropriate options, such as the enzyme used in the experiment. The help can be displayed using:

```
hicstuff pipeline --help
```

Matrices generated in the default coordinate format can then be visualised using the view command, modified using the rebin and convert commands, or used as input for other softwares.

## 1.3 hicstuff command line interface demo

### 1.3.1 Getting started

The easiest way to generate matrices is to use `hicstuff pipeline`. By default the command only requires two fastq files (forward and reverse reads) and a genome in fasta format.

The pipeline command can be used to generate the Hi-C contact map from the input reads.

```
hicstuff pipeline --genome genome.fa \
                  --outdir results \
                  forward.fq \
                  reverse.fq
```

For instance, this will create a directory named "results", containing three output text files with tab-separated columns.

- `abs_fragments_contacts_weighter.txt`: Sparse matrix file with 3 columns the rows, column and values of nonzero pixels. The first row contains the shape and total number of nonzero pixels in the matrix.
- `fragments_list.txt`: Contains genomic coordinates of the matrix bins (row/columns).
- `info_contigs.txt`: Contains chromosome names, theirs length and number of bins.

#### Using an indexed genome

When given a genome in fasta format, hicstuff regenerates the index everytime unless it finds an index starting by the genome filename. For example using bowtie2, you can index the genome like this:

```
bowtie2-build genome.fa genome.fa
```

And when running hicstuff with `--genome=genome.fa`, it will automatically find the index files and not regenerate it.

#### Setting the binning

By default, matrices generated are binned at 5kb. You can change this using the `--enzyme` option. This option allows to specify either a bin size, or an enzyme. For example if you set `--enzyme='DpnII'`, the matrix will be at restriction fragments resolution.

#### Temporary files

By default temporary files are removed when the pipeline finishes. They can be kept by adding the `--no-cleanup` flag. For example, if you run:

```
hicstuff pipeline -g genome.fa --no-cleanup --prefix demo
```

The `--prefix` option used here gives a common prefix to all output files, overriding the default output file names as follows:

- `abs_fragments_contacts_weighted.txt` -> `demo.mat.tsv`
- `fragments_list.txt` -> `demo.frags.tsv`
- `info_contigs.txt` -> `demo.chr.tsv`

And the output folder should now contain a `tmp/` subfolder and look like this:

```
output
├── demo.chr.tsv
├── demo.frags.tsv
├── demo.hicstuff_20190423185220.log
├── demo.mat.tsv
├── demo.distance_law.txt
├── plots
│   ├── event_distance.pdf
│   ├── event_distribution.pdf
│   └── frags_hist.pdf
└── tmp
    ├── demo.for.bam
    ├── demo.genome.fasta
    ├── demo.rev.bam
    ├── demo.valid_idx_filtered.pairs
    ├── demo.valid_idx.pairs
    └── demo.valid.pairs
```

#### Additional options

The `hicstuff pipeline` command has additional options allowing to tweak various parameters and change the output files or their format. You can always consult `hicstuff pipeline --help` to get a comprehensive description of those options, but here are a few of them:

- `--filter`: Filters out spurious 3C events, such as self religations or undigested fragments. This is only really useful at very fine resolutions (1-2kb) and not needed most of the time. This option is only meaningful when `--enzyme` is given a restriction enzyme and not a bin size.
- `--duplicates`: Removes PCR duplicates, defined as sets of pairs having identical mapping positions for both reads.
- `--matfmt`: allows to specify what file format should be used for the matrix. Available formats are bg2 (bedgraph2d), graal (the text format described above) and cool, a binary format that is probably the most appropriate for large genomes.
- `--distance-law`: Computes the distance-law table (i.e. the probability of contacts as a function of genomic distance).
- `--plot`: Enables plotting. When used in conjunction with `--filter` or `--distance-law`, this will generate figures showing properties of your Hi-C data.

### 1.3.2 Advanced usage

#### Starting from intermediate files

If your hicstuff run was interrupted, or you wish to align the reads separately, the `--start-stage` option allows the `hicstuff pipeline` command to take intermediate files as input. For example to skip the alignment step and start from aligned reads:

```
hicstuff pipeline --start-stage bam --genome genome.fa forward.bam reverse.bam
```

Or if you already have a pairs file:

```
hicstuff pipeline --start-stage pairs --genome genome.fa valid.pairs
```

### 1.3.3 Generating the distance law

The distance law is the probability of contact of two fragments in function of the distance between these fragments. There are two ways to compute it with hicstuff. The first one using the full pipeline with the option `--distance-law`, as done above. It's possible to add an option `--centromeres` if you want to compute the distance law on separate arms. The output of this command will be a raw table of the distance without any treatment of the data. It will be then possible with the command distancelaw to process this table.

The second way is to use the command distancelaw with the pairs file as input:

```
hicstuff distancelaw --average \
                     --big-arm-only \
                     --centromeres centromeres.txt \
                     --frags output/demo.frags.tsv \
                     --inf 3000 \
                     --outputfile-img output/demo_distance_law.svg \
                     --labels labels.txt \
                     --sup 500000 \
                     --pairs output/tmp/demo.valid_idx_filtered.pairs
```

For instance, this will create an image with the distance law generated from the pairs file given in input. The distance law will be the average between all the distance laws of the arms bigger than 500kb. The logspace used to plot it will have a base 1.1 by default. The limits of the x axis will be 3kb and 500kb.

Note that if `hicstuff pipeline` was given the `--distance-law` option, the output folder should contain a file named `distance_law.txt` containing the precomputed interaction frequencies. This file can be provided to the `hicstuff distancelaw` command using `--dist-tbl distance_law.txt` instead of the pairs file.

### 1.3.4 Operations on Hi-C matrices

All commands described below can take the output of hicstuff as input. They will accept either a bg2 matrix, a cool matrix or a graal matrix. Note that when using a graal matrix, you will usually need to specify the fragments file using `--frags` since genomic coordinates are not encoded in this matrix format.

#### Visualizing the matrix

Below are example commands that can be used to visualise Hi-C matrices with hicstuff.

```
# Viewing a normalized (=balanced) matrix in graal format at 5kb resolution
hicstuff view --binning 5kb --normalize --frags output/demo.frags.tsv output/demo.mat.
↪tsv
# Viewing a log ratio of 2 matrices in cool format
hicstuff view sample1.cool sample2.cool
# Viewing a raw matrix in bedgraph2 format at 500bp resolution with log transformed␣
↪contacts
hicstuff view --binning 500bp --transform log high_res.bg2
```

This will show an interactive heatmap using matplotlib. In order to save the matrix to a file instead, one could add `--output output/demo.png`

Note there are others options allowing to process the matrix which are documented in the help message.

### Converting between formats

Output files from `hicstuff pipeline` can be converted between different format using the command `hicstuff convert`. For example to generate the file `cool_output/demo.cool` from files in the default hicstuff format:

```
hicstuff convert --frags output/demo.frags.tsv \
                 --chroms output/demo.chr.tsv \
                 --to cool \
                 output/demo.mat.tsv
                 output/demo
```

Notice that the command takes 2 positional arguments, the first one is the matrix file and the second is the prefix to give for output files, to which the extension will be added depending on the output format chosen. Input format is inferred automatically from the input matrix.

### Rebinning existing matrices

Files previously produced by hicstuff pipeline can be rebinned at a lower resolutions using the `hicstuff rebin` command. This will generate a new matrix, a new fragments_list.txt and a new info_contigs.txt, all with updated number of bins:

```
hicstuff rebin -f output/demo.frags.tsv \
               -c output/demo.chr.tsv \
               --binning 1kb \
               output/demo.mat.tsv \
               rebin_1kb
```

When working with cool or bedgraph2 files, the command is simpler as we don't need fragments and contig files:

```
# Rebins demo.cool at 10kb and saves the results to rebinned.cool
hicstuff rebin --binning 10kb demo.cool rebinned
```

### Subsampling contacts

For many applications, differences in sequencing coverage will impact results. To avoid this, one can subsample contacts from Hi-C matrices to ensure the different samples to be compared have comparable signal. This functionality is implemented in the `hicstuff subsample` command, which allows to keep a fixed number of contacts from a matrix, or to extract a fraction of contacts:

```
# Keep 30% contacts in matrix.cool and save the results to subsamples_30.cool
hicstuff subsample --prop 0.5 matrix.cool subsample_30
# Keep 1 million contacts in matrix.bg2 and save the result in subsample_1M.bg2
hicstuff subsample --prop 1000000 matrix.bg2 subsample_1M
```

## 1.4 hicstuff API demo

All steps of hicstuff pipeline and downstream operations can be ran using the python api. The matrix can be generated directly from the reads, just like in the command line or by doing each step separately for fine control.

### 1.4.1 Data preparation

If using minimap2, the genome can be in fasta format. If using bowtie2, it must be indexed using bowtie2-build:

```
bowtie2-build genome.fasta genome
```

The input reads can be in fastq format, or if already aligned to the genome in SAM/BAM format. The pipeline also accepts input in the form of pairs file. The input format is specified using the `start_stage` argument (fastq/sam/pairs/pairs_idx).

### 1.4.2 Full pipeline using the API

The hicstuff.pipeline submodule allows to run all steps at one, in a way identical to the hicstuff pipeline command. Here we assume the reads have been aligned and converted to a pairs file already by specifying `start_stage='pairs'`, but it is also possible to give fastq files or sam files as input by specifying `start_stage='fastq'` or `start_stage='sam'`.

[204]:
```
import hicstuff.pipeline as hpi

# Generating matrix from dummy dataset
hpi.full_pipeline(input1='../../test_data/valid.pairs',
                  input2=None,
                  genome='../../test_data/genome/seq',
                  enzyme="DpnII",
                  filter_events=True,
                  pcr_duplicates=True,
                  start_stage='pairs')
```

```
INFO :: ## hicstuff: v1.4.4 log file
INFO :: ## date: 2019-07-17 16:48:36
INFO :: ## enzyme: DpnII
INFO :: ## input1: ../../test_data/valid.pairs
INFO :: ## input2: None
INFO :: ## ref: ../../test_data/genome/seq
INFO :: ---
INFO :: Filtering with thresholds: uncuts=6 loops=5
INFO :: Proportion of inter contacts: 1.97% (intra: 298, inter: 6)
INFO :: 9696 pairs discarded: Loops: 31, Uncuts: 9663, Weirds: 2
INFO :: 304 pairs kept (3.04%)
INFO :: 0% PCR duplicates have been filtered out (0 / 304 pairs)
INFO :: 304 pairs used to build a contact map of 564 bins with 304 nonzero entries.
INFO :: Contact map generated after 0h 0m 0s
```

### 1.4.3 Visualise output files

The temporary pairs files can be used to visualise the probability of contacts over genomic distance (distance law). Functions in the view module can also be used to generate visualisations of the results. The utilities in the hicstuff module are handy to transform the matrix.

[8]:
```
%matplotlib notebook
from importlib import reload
import hicstuff.distance_law as hdl
from matplotlib import pyplot as plt
reload(hdl)
```

(continues on next page)

(continued from previous page)

```
xs, ps, names = hdl.get_distance_law('example_data/example_clean.pairs', 'example_
↪data/fragments_list.txt')
# Normalize P(s) to sum to 1 between
ps = hdl.normalize_distance_law(xs, ps)
# Extract first chromosome only
chrom_xs, chrom_ps, chrom_name = xs[0], ps[0], names[0]
# Select interactions between 10 and 500kb
bp_range = (chrom_xs > 10000) & (chrom_xs < 500000)
# Plot log probability of contact versus log genomic distance
plt.loglog(chrom_xs[bp_range], chrom_ps[bp_range])
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f302008c550>]
```

Common operations for processing Hi-C matrices are made available in hicstuff. These operations work on scipy sparse matrices.

[6]:
```
%matplotlib notebook
from hicstuff import hicstuff as hcs
import hicstuff.view as hcv
import hicstuff.io as hio
from scipy.sparse import csr_matrix
from scipy.ndimage import gaussian_filter
import numpy as np
# Load COO sparse matrix from text file
sparse_mat = hio.load_sparse_matrix('example_data/abs_fragments_weighted.txt')
# Normalize matrix so that each genomic bin is equally covered
norm_mat = hcs.normalize_sparse(sparse_mat)
# Remove outlier bins with very little contacts
trim_mat = hcs.trim_sparse(norm_mat)
# Remove "speckles" (outlier pixel values) to reduce noise
clean_mat = hcs.despeckle_simple(trim_mat)
# Show the matrix
dense_mat = hcv.sparse_to_dense(clean_mat, remove_diag=False)
# Add some blur to the matrix :)
blurred_mat = gaussian_filter(dense_mat, 1.1)
# Plot blurred matrix
hcv.plot_matrix(blurred_mat, vmax=0.05)
```

```
WARNING :: /home/varogh/.local/lib/python3.6/site-packages/scipy/sparse/compressed.py:
↪708: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matrix is␣
↪expensive. lil_matrix is more efficient.
  self[i, j] = values
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[13]:
```
%matplotlib notebook
from matplotlib import pyplot as plt

# Visualize matrix at each different steps
todense = lambda x: hcv.sparse_to_dense(x, remove_diag=False)
mats = [
    ('raw',         todense(sparse_mat)),
```

(continues on next page)

(continued from previous page)

```
    ('normalized',  todense(norm_mat)),
    ('trimmed',     todense(trim_mat)),
    ('despeckled',  todense(clean_mat)),
    ('Gaussian blur', blurred_mat),
    ('shrec',       hcs.shortest_path_interpolation(dense_mat, alpha=0.04,␣
↪strict=True))
]

fig, ax = plt.subplots(3, 2, sharex=True, sharey=True, figsize=(12, 18), dpi=80)
for i, axi in enumerate(ax.flatten()):
    axi.set_aspect(1)
    axi.set_title(mats[i][0])
    axi.imshow(mats[i][1], cmap="Oranges", vmax=np.percentile(mats[i][1], 98.5))

plt.suptitle("Hi-C matrix at different processing steps")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[13]:
```
Text(0.5, 0.98, 'Hi-C matrix at different processing steps')
```

### Rebinning

It is also possible to rebin an existing matrix to a lower resolution. There are two types of binning in hicstuff: Base-pair binning where bins are of fixed genomic size, and subsample binning where new bins are formed by grouping N bins together. Basepair binning can be performed using `hicstuff.bin_bp_sparse`, while subsample binning uses `hicstuff.bin_sparse`.

[18]:
```
# Load COO sparse matrix at 1kb resolution
sparse_mat_1kb = hio.load_sparse_matrix('example_data/abs_fragments_weighted.txt')

# Rebin by subsampling groups of 5 fragments
mat_sub_5 = hcs.bin_sparse(sparse_mat_1kb, 5)

# Rebin by basepairs at 5kb. Genomic positions of fragments are required for this !
# Load start positions of fragments from fragments_list file
frags_start = hio.load_pos_col('example_data/fragments_list.txt', 2)
mat_5kb, frags_5kb = hcs.bin_bp_sparse(sparse_mat_1kb, frags_start, 5000)
```

[19]:
```
# Plotting matrix with both binning strategies
hcv.plot_matrix(hcv.sparse_to_dense(mat_5kb, remove_diag=False))
hcv.plot_matrix(hcv.sparse_to_dense(mat_sub_5, remove_diag=False))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

### Exporting

Processed matrices can be exported form python to a file.

[21]:
```
# Export sparse matrix
hio.save_sparse_matrix(mat_5kb, '5kb_matrix.txt')

# Export dense matrix
dense_5kb = hcv.sparse_to_dense(mat_5kb, remove_diag=False)
np.savetxt('5kb_dense_matrix.txt', dense_5kb)
```

### 1.4.4 Playing with formats

The contact map and fragments table can be used to generate a cool file using the `hicstuff.io.save_cool` command. Symmetrically, the `hicstuff.io.load_cool` command can be used to load the matrix and fragments list from a cool file.

[202]:
```
import pandas as pd
frags = pd.read_csv('example_data/fragments_list.txt', sep='\t')

hio.save_cool('example.cool', sparse_mat, frags)
```

```
INFO :: Creating cooler at "example.cool::/"
INFO :: Writing chroms
INFO :: Writing bins
INFO :: Writing pixels
INFO :: Writing indexes
INFO :: Writing info
INFO :: Done
```

CHAPTER 2

# Reference API

## 2.1 hicstuff package

### 2.1.1 Submodules

### 2.1.2 hicstuff.commands module

Abstract command classes for hicstuff

This module contains all classes related to hicstuff commands:

> -iteralign (iterative mapping) -digest (genome chunking) -cutsite (preprocess fastq by cutting reads into digestion products) -filter (Hi-C 'event' sorting: loops, uncuts, weird
>
> > and 'true contacts')
>
> -view (map visualization) -pipeline (whole contact map generation) -distancelaw (Analysis tool and plot for the distance law)

Running 'pipeline' implies running 'digest', but not iteralign or filter unless specified, because they can take up a lot of time for dimnishing returns.

---

**Note:** Structure based on Rémy Greinhofer (rgreinho) tutorial on subcommands in docopt : https://github.com/rgreinho/docopt-subcommands-example cmdoret, 20181412

---

**raises**

- `NotImplementedError` – Will be raised if AbstractCommand is called for some reason instead of one of its children.
- `ValueError` – Will be raised if an incorrect chunking method (e.g. not an enzyme or number or invalid range view is specified.

**class** hicstuff.commands.**AbstractCommand**(*command_args*, *global_args*)

Bases: `object`

Abstract base command class

Base class for the commands from which other hicstuff commands derive.

**check_output_path**(*path*, *force=False*)
Throws error if the output file exists. Create required file tree otherwise.

**execute**()
Execute the commands

**class** hicstuff.commands.**Convert**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Convert between different Hi-C dataformats. Currently supports tsv (graal), bedgraph2D (bg2) and cooler (cool). Input format is automatically inferred.

**usage:**

**convert [–frags=FILE] [–chroms=FILE] [–force] [–genome=FILE]** [–to=cool] <contact_map> <prefix>

**Parameters**

- **The file containing the contact frequencies.** (*contact_map*) –
- **The prefix path for output files. An extension** (*prefix*) – will be added to the files depending on the output format.

**options:**

| | |
|---|---|
| **-f, --frags=FILE** | Tab-separated file with headers, containing columns id, chrom, start_pos, end_pos size. This is the file "fragments_list.txt" generated by hicstuff pipeline. Required for graal matrices and recommended for bg2. |
| **-F, --force** | Write even if the output file already exists. |
| **-g, --genome=FILE** | Optional genome file used to add a GC content column to the fragments table. This is required to generate instagraal-compatible files. |
| **-c, --chroms=FILE** | Tab-separated with headers, containing columns contig, length, n_frags, cumul_length. This is the file "info_contigs.txt" generated by hicstuff pipeline. |
| **-T, --to=STR** | The format to which files should be converted. [default: cool] |

**execute**()
Execute the commands

**class** hicstuff.commands.**Cutsite**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Cutsite command

Generates new gzipped fastq files from original fastq. The function will cut the reads at their religation sites and creates new pairs of reads with the different fragments obtained after cutting at the digestion sites.

There are three choices to how combine the fragments. 1. "for_vs_rev": All the combinations are made between one forward fragment and one reverse fragment. 2. "all": All 2-combinations are made. 3. "pile": Only combinations between adjacent fragments in the initial reads are made.

**usage:** cutsite –forward=FILE –reverse=FILE –prefix=STR –enzyme=STR [–mode=for_vs_rev] [–seed-size=20] [–threads=1]

**options:**

| | |
|---|---|
| **-1, --forward=FILE** | Fastq file containing the forward reads to digest. |
| **-2, --reverse=FILE** | Fastq file containing the reverse reads to digest. |
| **-p, --prefix=STR** | Prefix of the path where to write the digested gzipped fastq files. Filenames will be added the suffix "_{1,2}.fq.gz". |
| **-e, --enzyme=STR** | The list of restriction enzyme used to digest the genome separated by a comma. Example: HpaII,MluCI. |
| **-m, --mode=STR** | Digestion mode. There are three possibilities: "for_vs_rev", "all" and "pile". The first one "for_vs_rev" makes all possible contact between fragments from forward read versus the fragments of the reverse reads. The second one "all" consist two make all pairs of fragments possible. The third one "pile" will make the contacts only with the adjacent fragments. [Default: for_vs_rev] |
| **-s, --seed-size=INT** | Minimum size of a read. (i.e. seed size used in mapping as reads smaller won't be mapped.) [Default: 20] |
| **-t, --threads=INT** | Number of parallel threads allocated for the alignement. [Default: 1] |

**execute**()
Execute the commands

**class** hicstuff.commands.**Digest**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Genome chunking command

Digests a fasta file into fragments based on a restriction enzyme or a fixed chunk size. Generates two output files into the target directory named "info_contigs.txt" and "fragments_list.txt"

**usage:**

**digest [–plot] [–figdir=FILE] [–force] [–circular] [–size=0]** [–outdir=DIR] –enzyme=ENZ <fasta>

**Parameters** **Fasta file to be digested**(*fasta*) –

**options:**

| | |
|---|---|
| **-c, --circular** | Specify if the genome is circular. |

**-e, –enzyme=ENZ[,ENZ2,...] A restriction enzyme or an integer** representing fixed chunk sizes (in bp). Multiple comma-separated enzymes can be given.

| | |
|---|---|
| **-F, --force** | Write even if the output file already exists. |
| **-s, --size=INT** | Minimum size threshold to keep fragments. [default: 0] |
| **-o, --outdir=DIR** | Directory where the fragments and contigs files will be written. Defaults to current directory. |
| **-p, --plot** | Show a histogram of fragment length distribution after digestion. |
| **-f, --figdir=FILE** | Path to directory of the output figure. By default, the figure is only shown but not saved. |

**output:** fragments_list.txt: information about restriction fragments (or chunks) info_contigs.txt: information about contigs or chromosomes

**execute**()
Execute the commands

**class** hicstuff.commands.**Distancelaw**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Distance law tools. Take the distance law file from hicstuff and can average it, normalize it compute the slope of the curve and plot it.

**usage:**

**distancelaw [–average] [–big-arm-only=INT] [–base=1.1] [–centromeres=FILE]** [–circular] [–frags=FILE] [–inf=INT] [–outputfile-img=FILE] [–outputfile-tabl=FILE] [–labels=DIR] [–sup=INT] [–remove-centromeres=0] (–pairs=FILE | –dist-tbl=FILE1[,FILE2,...])

**options:**

**-a, --average** If set, calculate the average of the distance law of the different chromosomes/arms in each condition. If two file given average is mandatory.

**-b, --big-arm-only=INT** Integer. It given will take only the arms bigger than the value given as argument.

**-B, --base=FLOAT** Float corresponding of the base of the log to make the logspace which will slice the genomes in logbins. These slices will be in basepairs unit. [default is 1.1]

**-c, --centromeres=FILE** Positions of the centromeres separated by a space and in the same order as the chromosomes. This allows to plot chromosomal arms separately. Note this will only work with –pairs input, as the distance law needs to be recomputed. Incompatible with the circular option.

**-C, --circular** Enable if the genome is circular. Discordant with the centromeres option.

**-d, –dist-tbl=FILE1[,FILE2,...] Directory to the file or files containing the** compute distance law. File should have the same format than the ones made by hicstuff pipeline.

**-f, --frags=FILE** Tab-separated file with headers, containing columns id, chrom, start_pos, end_pos size. This is the file "fragments_list.txt" generated by hicstuff pipeline. Required if pairs are given.

**-i, --inf=INT** Inferior born to plot the distance law. By default the value is 3000 bp (3 kb). Have to be strictly positive.

**-l, –labels=STR1,STR2... List of string of the labels for the plot** separated by a coma. If no labels given, give the names "Sample 1", "Sample 2"...

**-o, --outputfile-img=FILE** Output file. Format must be compatible with plt.savefig. Default : None.

**-O, --outputfile-tabl=TABLE** Output file. Default : None.

**-p, --pairs=FILE** Pairs file. Format from 4D Nucleome Omics Data Standards Working Group with the 8th and 9th coulumns are the ID of the fragments of the reads 1 and 2. Only add if no distance_law table given. It will compute the table from these pairs and the fragments from the fragments file.

**-r, --remove-centromeres=INT** Integer. Number of kb that will be remove around the centromere position given by in the centromere file. [default: 0]

**-s, --sup=INT** Superior born to plot the distance law. By default the value is the maximum length of all the dataset given. Also if big arm only set, it will be the minimum size of the arms/chromosomes taken to make the average.

**execute**()
Execute the commands

**class** hicstuff.commands.**Filter**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Mapping event filtering command

Filters spurious 3C events such as loops and uncuts from the library based on a minimum distance threshold automatically estimated from the library by default. Can also plot 3C library statistics.

**usage:**

**filter [–interactive | –thresholds INT-INT] [–plot]** [–figdir FILE] [–prefix STR] <input> <output>

**Parameters**

- **2D BED file containing coordinates of Hi-C interacting** (*input*) – pairs, the index of their restriction fragment and their strands.
- **Path to the filtered file, in the same format as the input.** (*output*) –

**options:**

**-f, --figdir=DIR** Path to the output figure directory. By default, the figure is only shown but not saved.

**-i, --interactive** Interactively shows plots and asks for thresholds.

**-p, --plot** Shows plots of library composition and 3C events abundance.

**-P, --prefix STR** If the library has a name, it will be shown on the figures.

**-t, --thresholds=INT-INT** Manually defines integer values for the thresholds in the order [uncut, loop]. Reads above those values are kept.

**execute**()
Execute the commands

**class** hicstuff.commands.**Iteralign**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Iterative mapping command

Truncate reads from a fastq file to 20 basepairs and iteratively extend and re-align the unmapped reads to optimize the proportion of uniquely aligned reads in a 3C library.

**usage:**

**iteralign [–aligner=bowtie2] [–threads=1] [–min-len=20] [–read-len=INT]** [–tempdir=DIR] –out-bam=FILE –genome=FILE <reads.fq>

**Parameters** **Fastq file containing the reads to be aligned**(*reads.fq*) –

**options:**

**-g, --genome=FILE** The genome on which to map the reads. Must be the path to the bowtie2/bwa index if using bowtie2/bwa or to the genome in fasta format if using minimap2.

**-t, --threads=INT** Number of parallel threads allocated for the alignment [default: 1].

**-T, --tempdir=DIR** Temporary directory. Defaults to current directory.

**-a, --aligner=bowtie2** Choose alignment software between bowtie2, minimap2 or bwa. minimap2 should only be used for reads > 100 bp. [default: bowtie2]

**-l, --min-len=INT** Length to which the reads should be truncated [default: 20].

**-o, --out-bam=FILE** Path where the alignment will be written in BAM format.

**-R, --read-len=INT** Read length in input FASTQ file. If not provided, this is estimated from the first read in the file.

**execute**()
Execute the commands

**class** hicstuff.commands.**Missview**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Previews bins that will be missing in a Hi-C map with a given read length by finding repetitive regions in the genome.

**usage:**

**missview [–aligner=bowtie2] [–force] [–binning=5000]** [–threads=1] [–tmpdir=STR] –read-len=INT <genome> <output>

**Parameters**

- **Genome file in fasta format.** (*genome*) –
- **Path to the output image.** (*output*) –

**options:**

**-a, --aligner=STR** The read alignment software to use. Can be either bowtie2, minimap2 or bwa. minimap2 should only be used for reads > 100 bp. [default: bowtie2]

**-b, --binning=INT** Resolution to use to preview the Hi-C map. [default: 5000]

**-F, --force** Write even if the output file already exists.

**-R, --read-len=INT** Write even if the output file already exists.

**-t, --threads=INT** Number of CPUs to use in parallel. [default: 1]

**-T, --tmpdir=STR** Directory where temporary files will be generated.

**execute**()
Execute the commands

**class** hicstuff.commands.**Pipeline**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Whole (end-to-end) contact map generation command

Entire Pipeline to process fastq files into a Hi-C matrix. Uses all the individual components of hicstuff.

**usage:**

**pipeline [–aligner=bowtie2] [–centromeres=FILE] [–circular] [–distance-law]** [–duplicates] [–enzyme=5000] [–filter] [–force] [–mapping=normal] [–matfmt=graal] [–no-cleanup] [–outdir=DIR] [–plot] [–prefix=PREFIX] [–quality-min=30] [–read-len=INT] [–remove-centromeres=0] [–size=0] [–start-stage=fastq] [–threads=1] [–tmpdir=DIR] –genome=FILE <input1> [<input2>]

**Parameters**

- **input1** – Forward fastq file, if start_stage is "fastq", sam file for aligned forward reads if start_stage is "bam", or a .pairs file if start_stage is "pairs".
- **input2** – Reverse fastq file, if start_stage is "fastq", sam file for aligned reverse reads if start_stage is "bam", or nothing if start_stage is "pairs".

**options:**

**-a, --aligner=STR** Alignment software to use. Can be either bowtie2, minimap2 or bwa. minimap2 should only be used for reads > 100 bp. [default: bowtie2]

**-c, --centromeres=FILE** Positions of the centromeres separated by a space and in the same order than the chromosomes. Discordant with the circular option.

**-C, --circular** Enable if the genome is circular. Discordant with the centromeres option.

**-d, --distance-law** If enabled, generates a distance law file with the values of the probabilities to have a contact between two distances for each chromosomes or arms if the file with the positions has been given. The values are not normalized, or averaged.

**-D, --duplicates** Filter out PCR duplicates based on read positions.

**-e, –enzyme={STR|INT} Restriction enzyme or "mnase" if a string,** or chunk size (i.e. resolution) if a number. Can also be multiple comma-separated enzymes. [default: 5000]

**-f, --filter** Filter out spurious 3C events (loops and uncuts) using hicstuff filter. Requires "-e" to be a restriction enzyme or mnase, not a chunk size. For more informations, see Cournac et al. BMC Genomics, 2012.

**-F, --force** Write even if the output file already exists.

**-g, --genome=FILE** Reference genome to map against. Path to the bowtie2/bwa index if using bowtie2/bwa, or to a FASTA file if using minimap2.

**-m, --mapping=STR** normal|iterative|cutsite. Parameter of mapping. "normal": Directly map reads without any process. "iterative": Map reads iteratively using iteralign, by truncating reads to 20bp and then repeatedly extending to align them. "cutsite": Cut reads at the religation sites of the given enzyme using cutsite, create new pairs of reads and then align them ; enzyme is required [default: normal].

**-M, --matfmt=STR** The format of the output sparse matrix. Can be "bg2" for 2D Bedgraph format, "cool" for Mirnylab's cooler software, or "graal" for graal-compatible plain text COO format. [default: graal]

**-n, --no-cleanup** If enabled, intermediary BED files will be kept after generating the contact map. Disabled by defaut.

**-o, --outdir=DIR** Output directory. Defaults to the current directory.

**-p, --plot** Generates plots in the output directory at different steps of the pipeline.

**-P, --prefix=STR** Overrides default filenames and prefixes all output files with a custom name.

**-q, --quality-min=INT** Minimum mapping quality for selecting contacts. [default: 30].

**-r, --remove-centromeres=INT** Integer. Number of kb that will be remove around the centromere position given by in the centromere file. [default: 0]

**-R, --read-len=INT** Maximum read length in the fastq file. Optionally used in iterative alignment mode. Estimated from the first read by default. Useful if input fastq is a composite of different read lengths.

**-s, --size=INT** Minimum size threshold to consider contigs. Keep all contigs by default. [default: 0]

**-S, --start-stage=STR** Define the starting point of the pipeline to skip some steps. Default is "fastq" to run from the start. Can also be "bam" to skip the alignment, "pairs" to start from a single pairs file or "pairs_idx" to skip fragment attribution and only build the matrix. [default: fastq]

**-t, --threads=INT** Number of threads to allocate. [default: 1].

**-T, --tmpdir=DIR** Directory for storing intermediary BED files and temporary sort files. Defaults to the output directory.

**output:** abs_fragments_contacts_weighted.txt: the sparse contact map fragments_list.txt: information about restriction fragments (or chunks) info_contigs.txt: information about contigs or chromosomes hicstuff.log: details and statistics about the run.

**execute**()
Execute the commands

**class** hicstuff.commands.**Rebin**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Rebins a Hi-C matrix and modifies its fragment and chrom files accordingly. Output files are in the same format as the input files (cool, graal or bg2). usage:

rebin [–binning=1] [–frags=FILE] [–force] [–chroms=FILE] <contact_map> <out_prefix>

**Parameters**

- **Sparse contact matrix in graal, cool or bg2 format.** (*contact_map*) –
- **Prefix path** (*out_prefix*) –

**options:**

**-b, –binning=INT[bp|kb|Mb|Gb] Subsampling factor or fix value in** basepairs to use for binning [default: 1].

**-f, --frags=FILE** Tab-separated file with headers, containing fragments start position in the 3rd column. This is the file "fragments_list.txt" generated by hicstuff pipeline. Required for graal matrices and recommended for bg2.

**-F, --force** Write even if the output file already exists.

**-c, --chroms=FILE** Tab-separated with headers, containing chromosome names, size, number of restriction fragments. This is the file "info_contigs.txt" generated by hicstuff pipeline.

**execute**()
Execute the commands

**class** hicstuff.commands.**Scalogram**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Generate a scalogram.

**usage:**

**scalogram [–cmap=viridis] [–centromeres=FILE] [–frags=FILE] [–range=INT-INT]** [–threads=1] [–output=FILE] [–normalize] [–indices=INT-INT] [–despeckle] <contact_map>

**argument:** <contact_map> The sparse Hi-C contact matrix.

**options:**

**-C, --cmap=STR** The matplotlib colormap to use for the plot. [default: viridis]

**-d, --despeckle** Remove speckles (artifactual spots) from the matrix.

**-f, --frags=FILE** Fragments_list.txt file providing mapping between genomic coordinates and bin IDs.

**-i, --indices=INT-INT** The range of bin numbers of the matrix to use for the plot. Can also be given in UCSC style genomic coordinates (requires -f). E.g. chr1:1Mb-10Mb.

**-o, --output=FILE** Output file where the plot should be saved. Plot is only displayed by default.

**-n, --normalize** Normalize the matrix first.

**-r, --range=INT-INT** The range of contact distance to look at. No limit by default. Values in basepairs by default but a unit can be specified (kb, Mb, . . . ).

**-t, --threads=INT** Parallel processes to run in for despeckling. [default: 1]

**execute**()
Execute the commands

**class** hicstuff.commands.**Subsample**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Subsample contacts from a Hi-C matrix. Probability of sampling is proportional to the intensity of the bin. usage:

subsample [–prop=0.1] [–force] <contact_map> <subsampled_prefix>

**Parameters**

- **Sparse contact matrix in graal, bg2 or cool format.** (*contact_map*) –
- **Path without extension to the output map in the same** (*subsampled_prefix*) – format as the input containing only a fraction of the contacts.

**options:**

**-p, --prop=FLOAT** Proportion of contacts to sample from the input matrix if between 0 and 1. Raw number of contacts to keep if superior to 1. [default: 0.1]

**-F, --force** Write even if the output file already exists.

**execute**()
Execute the commands

**class** hicstuff.commands.**View**(*command_args*, *global_args*)
Bases: *hicstuff.commands.AbstractCommand*

Contact map visualization command

Visualize a Hi-C matrix file as a heatmap of contact frequencies. Allows to tune visualisation by binning and normalizing the matrix, and to save the output image to disk. If no output is specified, the output is displayed.

**usage:**

**view [–binning=1] [–despeckle] [–frags FILE] [–trim INT] [–n-mad 3.0] [–lines]** [–normalize] [–min=0] [–max=99%] [–output=IMG] [–cmap=Reds] [–dpi=300] [–transform=STR] [–circular] [–region=STR] <contact_map> [<contact_map2>]

**Parameters**

- **Sparse contact matrix in bg2, cool or graal format** (*contact_map*) –
- **Sparse contact matrix in bg2, cool or graal format,** (*contact_map2*) – if given, the log ratio of contact_map/contact_map2 will be shown.

**options:**

**-b, –binning=INT[bp|kb|Mb|Gb] Rebin the matrix. If no unit is given, bins will** be merged by groups of INT. If a unit is given, bins of that size will be generated. [default: 1]

**-c, --cmap=STR** The name of a matplotlib colormap to use for the matrix. [default: Reds]

**-C, --circular** Use if the genome is circular.

**-d, --despeckle** Remove sharp increases in long range contact by averaging surrounding values.

**-D, --dpi=INT** Map resolution in DPI (dots per inch). [default: 300]

**-f, --frags=FILE** Required for bp binning and chromosome lines. Tab-separated file with headers, containing fragments start position in the 3rd column, as generated by hicstuff pipeline.

**-T, --transform=STR** Apply a mathematical transformation to pixel values to improve visibility of long range signals. Possible values are: log2, log10, ln, sqrt, exp0.2.

**-l, --lines** Add dotted lines marking separation between chromosomes or contigs. Requires –frags.

**-M, --max=INT** Saturation threshold. Maximum pixel value is set to this number. Can be followed by % to use a percentile of nonzero pixels in the contact map. [default: 99%]

**-m, --min=INT** Minimum of the colorscale, works identically to –max. [default: 0]

**-N, --n-mad=INT** Number of median absolute deviations (MAD) from the median of log bin sums allowed to keep bins in the normalization procedure [default: 3.0].

**-n, --normalize** Should ICE normalization be performed before rendering the matrix ?

**-o, --output=FILE** Name of the image file where the view is stored.

**-r, –region=STR[;STR] Only view a region of the contact map.** Regions are specified as UCSC strings. (e.g.:chr1:1000-12000). If only one region is given, it is viewed on the diagonal. If two regions are given, The contacts between both are shown.

**-t, --trim=INT** Trims outlier rows/columns from the matrix if the sum of their contacts deviates from the mean by more than INT standard deviations.

**data_transform**(*dense_map*, *operation='log10'*)
Apply a mathematical operation on a dense Hi-C map. Valid operations are: log2, log10, ln, sqrt, exp0.2

**execute**()
Execute the commands

**process_matrix**(*sparse_map*)
Performs any combination of binning, normalisation, log transformation, trimming and subsetting based on the attributes of the instance class.

hicstuff.commands.**parse_bin_str**(*bin_str*)
Bin string parsing

Take a basepair binning string as input and converts it into corresponding basepair values.

**Parameters** **bin_str** (*str*) – A basepair region (e.g. 150KB). Unit can be BP, KB, MB, GB.

### Example

```
>>> parse_bin_str("150KB")
150000
>>> parse_bin_str("0.1mb")
100000
```

**Returns** **binning** – The number of basepair corresponding to the binning string.

**Return type** int

hicstuff.commands.**parse_ucsc**(*ucsc_str*, *bins*)
Take a UCSC region in UCSC notation and a list of bin chromosomes and positions (in basepair) and converts it to range of bins.

**Parameters**

- **ucsc_str** (*str*) – The region string in UCSC notation (e.g. chr1:1000-2000)
- **bins** (*pandas.DataFrame*) – Dataframe of two columns containing the chromosome and start position of each bin. Each row must be one bin.

**Returns** **coord** – A tuple containing the bin range containing in the requested region.

**Return type** tuple

### 2.1.3 hicstuff.digest module

Genome digestion

Functions used to write auxiliary instagraal compatible sparse matrices.

hicstuff.digest.**attribute_fragments**(*pairs_file*, *idx_pairs_file*, *restriction_table*)

Writes the indexed pairs file, which has two more columns than the input pairs file corresponding to the restriction fragment index of each read. Note that pairs files have 1bp point positions whereas restriction table has 0bp point poisitions.

**Parameters**

- **pairs_file** (*str*) – Path the the input pairs file. Consists of 7 tab-separated columns: readID, chr1, pos1, chr2, pos2, strand1, strand2
- **idx_pairs_file** (*str*) – Path to the output indexed pairs file. Consists of 9 white space separated columns: readID, chr1, pos1, chr2, pos2, strand1, strand2, frag1, frag2. frag1 and frag2 are 0-based restriction fragments based on whole genome.
- **restriction_table** (*dict*) – Dictionary with chromosome identifiers (str) as keys and list of positions (int) of restriction sites as values.

hicstuff.digest.**find_frag**(*pos*, *r_sites*)

Use binary search to find the index of a chromosome restriction fragment corresponding to an input genomic position. :param pos: Genomic position, in base pairs. :type pos: int :param r_sites: List of genomic positions corresponding to restriction sites. :type r_sites: list

**Returns**

- *int* – The 0-based index of the restriction fragment to which the position belongs.
- >>> *find_frag(15, [0, 20, 30])*
- *0*
- >>> *find_frag(15, [10, 20, 30])*
- *Traceback (most recent call last)* – ...
- **ValueError** (*The first position in the restriction table is not 0.*)
- >>> *find_frag(31, [0, 20, 30])*
- *Traceback (most recent call last)* – ...
- **ValueError** (*Read position is larger than last entry in restriction table.*)

hicstuff.digest.**frag_len**(*frags_file_name='fragments_list.txt'*, *output_dir=None*, *plot=False*, *fig_path=None*)

logs summary statistics of fragment length distribution based on an input fragment file. Can optionally show a histogram instead of text summary. :param frags_file_name: Path to the output list of fragments. :type frags_file_name: str :param output_dir: Directory where the list should be saved. :type output_dir: str :param plot: Wether a histogram of fragment length should be shown. :type plot: bool :param fig_path: If a path is given, the figure will be saved instead of shown. :type fig_path: str

hicstuff.digest.**gen_enzyme_religation_regex**(*enzyme*)

Return a regex which corresponds to all possible religation sites given a set of enzyme. Parameters: ——— enzyme : str

String that contains the names of the enzyme separated by a comma.

**re.Pattern :** Regex that corresponds to all possible ligation sites given a set of enzyme.

```
>>> gen_enzyme_religation_regex('HpaII')
re.compile('CCGCGG')
>>> gen_enzyme_religation_regex('HpaII,MluCI')
re.compile('AATTAATT|AATTCGG|CCGAATT|CCGCGG')
```

hicstuff.digest.**get_restriction_table**(*seq*, *enzyme*, *circular=False*)

Get the restriction table for a single genomic sequence.

**Parameters**

- **seq** (*Seq object*) – A biopython Seq object representing a chromosomes or contig.
- **enzyme** (*int, str or list of str*) – The name of the restriction enzyme used, or a list of restriction enzyme names. Can also be an integer, to digest by fixed chunk size.
- **circular** (*bool*) – Wether the genome is circular.

**Returns**

- *numpy.array* – List of restriction fragment boundary positions for the input sequence.
- >>> *from Bio.Seq import Seq*
- >>> *get_restriction_table(Seq("AAGCCGGATCGG"),"HpaII")*
- *array([ 0, 4, 12])*
- >>> *get_restriction_table(Seq("AA"),["HpaII", "MluCI"])*
- *array([0, 2])*
- >>> *get_restriction_table(Seq("AA"),"aeiou1")*
- *Traceback (most recent call last) – ...*
- **ValueError** (*aeiou1 is not a valid restriction enzyme.*)
- >>> *get_restriction_table("AA","HpaII")*
- *Traceback (most recent call last) – ...*
- **TypeError** (*Expected Seq or MutableSeq instance, got <class 'str'> instead*)

hicstuff.digest.**write_frag_info**(*fasta*, *enzyme*, *min_size=0*, *circular=False*, *output_contigs='info_contigs.txt'*, *output_frags='fragments_list.txt'*, *output_dir=None*)

Digest and write fragment information

Write the fragments_list.txt and info_contigs.txt that are necessary for instagraal to run.

**Parameters**

- **fasta** (*pathlib.Path or str*) – The path to the reference genome
- **enzyme** (*str, int or list of str*) – If a string, must be the name of an enzyme (e.g. DpnII) and the genome will be cut at the enzyme's restriction sites. If a number, the genome will be cut uniformly into chunks with length equal to that number. A list of enzymes can also be specified if using multiple enzymes.
- **min_size** (*float, optional*) – Size below which shorter contigs are discarded. Default is 0, i.e. all contigs are retained.
- **circular** (*bool, optional*) – Whether the genome is circular. Default is False.
- **output_contigs** (*str, optional*) – The name of the file with contig info. Default is info_contigs.txt

- **output_frags** (*str, optional*) – The name of the file with fragment info. Default is fragments_list.txt
- **output_dir** (*[type], optional*) – The path to the output directory, which will be created if not already existing. Default is the current directory.

### 2.1.4 hicstuff.distance_law module

hicstuff.distance_law.**average_distance_law**(*xs*, *ps*, *sup*, *big_arm_only=False*)

Compute the average distance law between the file the different distance law of the chromosomes/arms.

**Parameters**

- **xs** (*list of numpy.ndarray*) – The list of logbins.
- **ps** (*list of lists of floats*) – The list of numpy.ndarray.
- **sup** (*int*) – Value given to set the minimum size of the chromosomes/arms to make the average.
- **big_arm_only** (*bool*) – By default False. If True, will only take into account the arms/chromosomes longer than the value of sup. Sup mandatory if set.

**Returns**

- *numpy.ndarray* – List of the xs with the max length.
- *numpy.ndarray* – List of the average_ps.

hicstuff.distance_law.**circular_distance_law**(*distance*, *chr_segment_length*, *chr_bin*)

Recalculate the distance to return the distance in a circular chromosome and not the distance between the two genomic positions.

**Parameters**

- **chr_segment_bins** (*list of floats*) – The start and end indices of chromosomes/arms to compute the distance law on each chromosome/arm separately.
- **chr_segment_length** (*list of floats*) – List of the size in base pairs of the different arms or chromosomes.
- **distance** (*int*) – Distance between two fragments with a contact.

**Returns** The real distance in the chromosome circular and not the distance between two genomic positions

**Return type** int

#### Examples

```
>>> circular_distance_law(7500, [2800, 9000], 1)
1500
>>> circular_distance_law(1300, [2800, 9000], 0)
1300
>>> circular_distance_law(1400, [2800, 9000], 0)
1400
```

hicstuff.distance_law.**export_distance_law**(*xs*, *ps*, *names*, *out_file=None*)

Export the x(s) and p(s) from two list of numpy.ndarrays to a table in txt file with three columns separated by a tabulation. The first column contains the x(s), the second the p(s) and the third the name of the arm or chromosome. The file is created in the directory given by outdir or the current file if no file is given.

**Parameters**

- **xs** (*list of numpy.ndarray*) – The list of the start position of logbins of each p(s) in base pairs.
- **ps** (*list of numpy.ndarray*) – The list of p(s).
- **names** (*list of string*) – List containing the names of the chromosomes/arms/conditions of the p(s) values given.
- **out_file** (*str or None*) – Path where output file should be written. ./distance_law.txt by default.

**Returns** File with three columns separated by a tabulation. The first column contains the x(s), the second the p(s) and the third the name of the arm or chromosome. The file is creates in the output file given or the default one if none given.

**Return type** txt file

hicstuff.distance_law.**get_chr_segment_bins_index**(*fragments*, *centro_file=None*, *rm_centro=0*)

Get the index positions of the start and end bins of different chromosomes, or arms if the centromers position have been given from the fragments file made by hicstuff.

**Parameters**

- **fragments** (*pandas.DataFrame*) – Table containing in the first coulum the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **centro_file** (*None or str*) – None or path to a file with the genomic positions of the centromers sorted as the chromosomes separated by a space. The file have only one line.
- **rm_centro** (*int*) – If a value is given, will remove the contacts close the centromeres. It will remove as many kb as the argument given. Default is zero.

**Returns** The start and end indices of chromosomes/arms to compute the distance law on each chromosome/arm separately.

**Return type** list of floats

hicstuff.distance_law.**get_chr_segment_length**(*fragments*, *chr_segment_bins*)

Compute a list of the length of the different objects (arm or chromosome) given by chr_segment_bins.

**Parameters**

- **fragments** (*pandas.DataFrame*) – Table containing in the first coulum the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **chr_segment_bins** (*list of floats*) – The start and end indices of chromosomes/arms to compute the distance law on each chromosome/arm separately.

**Returns** The length in base pairs of each chromosome or arm.

**Return type** list of numpy.ndarray

hicstuff.distance_law.**get_distance_law**(*pairs_reads_file*, *fragments_file*, *centro_file=None*, *base=1.1*, *out_file=None*, *circular=False*, *rm_centro=0*)

Compute distance law as a function of the genomic coordinate aka P(s). Bin length increases exponentially with distance. Works on pairs file format from 4D Nucleome Omics Data Standards Working Group. If the genome

is composed of several chromosomes and you want to compute the arms separately, provide a file with the positions of centromers. Create a file with three coulumns separated by a tabulation. The first column contains the xs, the second the ps and the third the name of the arm or chromosome. The file is create in the directory given in outdir or in the current directory if no directory given.

**Parameters**

- **pairs_reads_file** (*string*) – Path of a pairs file format from 4D Nucleome Omics Data Standards Working Group with the 8th and 9th coulumns are the ID of the fragments of the reads 1 and 2.
- **fragments_file** (*path*) – Path of a table containing in the first column the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **centro_file** (*None or str*) – None or path to a file with the genomic positions of the centromers sorted as the chromosomes separated by a space. The file have only one line.
- **base** (*float*) – Base use to construct the logspace of the bins - 1.1 by default.
- **out_file** (*None or str*) – Path of the output file. If no path given, the output is returned.
- **circular** (*bool*) – If True, calculate the distance as the chromosome is circular. Default value is False. Cannot be True if centro_file is not None
- **rm_centro** (*int*) – If a value is given, will remove the contacts close the centromeres. It will remove as many kb as the argument given. Default is None.

**Returns**

- **xs** (*list of numpy.ndarray*) – Basepair coordinates of log bins used to compute distance law.
- **ps** (*list of numpy.ndarray*) – Contacts value, in arbitrary units, at increasingly long genomic ranges given by xs.
- **names** (*list of strings*) – Names of chromosomes that are plotted

hicstuff.distance_law.**get_names**(*fragments*, *chr_segment_bins*)

Make a list of the names of the arms or the chromosomes.

**Parameters**

- **fragments** (*pandas.DataFrame*) – Table containing in the first coulum the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **chr_segment_bins** (*list of floats*) – The start position of chromosomes/arms to compute the distance law on each chromosome/arm separately.

**Returns** List of the labels given to the curves. It will be the name of the arms or chromosomes.

**Return type** list of floats

hicstuff.distance_law.**get_pairs_distance**(*line*, *fragments*, *chr_segment_bins*, *chr_segment_length*, *xs*, *ps*, *circular=False*)

From a line of a pair reads file, filter -/+ or +/- reads, keep only the reads in the same chromosome/arm and compute the distance of the the two fragments. It modify the input ps in order to count or not the line given. It will add one in the logbin corresponding to the distance.

**Parameters**

- **line** (*OrderedDict*) – Line of a pair reads file with the these keys readID, chr1, pos1, chr2, pos2, strand1, strand2, frag1, frag2. The values are in a dictionnary.
- **fragments** (*pandas.DataFrame*) – Table containing in the first coulum the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **chr_segment_bins** (*list of floats*) – The start and end indices of chromosomes/arms to compute the distance law on each chromosome/arm separately.
- **chr_segment_length** (*list of floats*) – List of the size in base pairs of the different arms or chromosomes.
- **xs** (*list of lists*) – The start coordinate of each bin one array per chromosome or arm.
- **ps** (*list of lists*) – The sum of contact already count. xs and ps should have the same dimensions.
- **circular** (*bool*) – If True, calculate the distance as the chromosome is circular. Default value is False.

hicstuff.distance_law.**get_ylim**(*xs*, *curve*, *inf*, *sup*)

Compute the max and the min of the list of list between the inferior (inf) and superior (sup) bounds.

**Parameters**

- **xs** (*list of numpy.ndarray*) – The list of the logbins starting position in basepair.
- **curve** (*list of numpy.ndarray*) – A list of numpy.ndarray from which you want to extract minimum and maximum values in a given interval.
- **inf** (*int*) – Inferior limit of the interval in basepair.
- **sup** (*int*) – Superior limit of the interval in basepair.

**Returns**

- **min_tot** (*float*) – Minimum value of the list of list in this interval.
- **max_tot** (*float*) – Maximum value of the list of list in this interval.

### Examples

```
>>> get_ylim([np.array([1, 4, 15]), np.array([1, 4, 15, 40])],
... [np.array([5.5, 3.2, 17.10]), np.array([24, 32, 1.111, 18.5])],
... 2,
... 15
... )
(1.111, 32.0)
```

hicstuff.distance_law.**import_distance_law**(*distance_law_file*)

Import the table created by export_distance_law and return the list of x(s) and p(s) in the order of the chromosomes.

**Parameters** **distance_law_file** (*string*) – Path to the file containing three columns : the x(s), the p(s), and the chromosome/arm name.

**Returns**

- *list of numpy.ndarray* – The start coordinate of each bin one array per chromosome or arm.

- *list of numpy.ndarray* – The distance law probabilities corresponding of the bins of the previous list.
- *list of numpy.ndarray* – The names of the arms/chromosomes corresponding to the previous list.

hicstuff.distance_law.**logbins_xs**(*fragments*, *chr_segment_length*, *base=1.1*, *circular=False*)

Compute the logbins of each chromosome/arm in order to have theme to compute distance law. At the end you will have bins of increasing with a logspace with the base of the value given in base.

**Parameters**

- **fragments** (*pandas.DataFrame*) – Table containing in the first coulum the ID of the fragment, in the second the names of the chromosome in the third and fourth the start position and the end position of the fragment. The file have no header. (File like the 'fragments_list.txt' from hicstuff)
- **chr_segment_length** (*list of floats*) – List of the size in base pairs of the different arms or chromosomes.
- **base** (*float*) – Base use to construct the logspace of the bins, 1.1 by default.
- **circular** (*bool*) – If True, calculate the distance as the chromosome is circular. Default value is False.

**Returns** The start coordinate of each bin one array per chromosome or arm.

**Return type** list of numpy.ndarray

hicstuff.distance_law.**normalize_distance_law**(*xs*, *ps*, *inf=3000*, *sup=None*)

Normalize the distance in order to have the sum of the ps values between 'inf' (default value is 3kb) until the end of the array equal to one and limit the effect of coverage between two conditions/chromosomes/arms when you compare them together. If we have a list of ps, it will normalize until the length of the shorter object or the value of sup, whichever is smaller.

**Parameters**

- **xs** (*list of numpy.ndarray*) – list of logbins corresponding to the ps.
- **ps** (*list of numpy.ndarray*) – Average ps or list of ps of the chromosomes/arms. xs and ps have to have the same shape.
- **inf** (*integer*) – Inferior value of the interval on which, the normalization is applied.
- **sup** (*integer*) – Superior value of the interval on which, the normalization is applied.

**Returns** List of ps each normalized separately.

**Return type** list of numpy.ndarray

hicstuff.distance_law.**plot_ps_slope**(*xs*, *ps*, *labels*, *fig_path=None*, *inf=3000*, *sup=None*)

Compute two plots, one with the different distance law of each arm/chromosome in loglog scale and one with the slope (derivative) of these curves. Generate a svg file with savefig.

**Parameters**

- **xs** (*list of numpy.ndarray*) – The list of the logbins of each ps.
- **ps** (*list of numpy.ndarray*) – The list of ps.
- **labels_file** (*list of strings*) – Names of the different curves in the order in which they are given.
- **fig_path** (*str*) – Path where the figure will be created. If None (default), the plot is shown in an interactive window.

- **inf** (*int*) – Value of the mimimum x of the window of the plot. Have to be strictly positive. By default 3000.
- **sup** (*int*) – Value of the maximum x of the window of the plot. By default None.

hicstuff.distance_law.**slope_distance_law**(*xs*, *ps*)

Compute the slope of the loglog curve of the ps as the [log(ps(n+1)) - log(ps(n))] / [log(n+1) - log(n)]. Compute only list of ps, not list of array.

**Parameters**

- **xs** (*list of numpy.ndarray*) – The list of logbins.
- **ps** (*list of numpy.ndarray*) – The list of ps.

**Returns** The slope of the distance law. It will be shorter of one value than the ps given initially.

**Return type** list of numpy.ndarray

### 2.1.5 hicstuff.filter module

Hi-C event filtering

Analyse the contents of a 3C library and filters spurious events such as loops and uncuts to improve the overall signal. Filtering consists in excluding +- and -+ Hi-C pairs if their reads are closer than a threshold in minimum number of restriction fragments. This threshold represents the distance at which the abundance of these events deviate significantly from the rest of the library. It is estimated automatically by default using the median absolute deviation of pairs at longer distances, but can also be entered manually.

The program takes a 2D BED file as input with the following fields: chromA startA endA indiceA strandA chromB startB endB indiceB strandB Each line of the file represents a Hi-C pair with reads A and B. The indices are 0-based and represent the restriction fragment to which reads are attributed.

@author: cmdoret (reimplementation of Axel KournaK's code)

hicstuff.filter.**filter_events**(*in_dat*, *out_filtered*, *thr_uncut*, *thr_loop*, *plot_events=False*, *fig_path=None*, *prefix=None*)

Filter events (loops, uncuts and weirds)

Filter out spurious intrachromosomal Hi-C pairs from input file. +- pairs with reads closer or at the uncut threshold and -+ pairs with reads closer or at the loop thresholds are excluded from the ouput file. – and ++ pairs with both mates on the same fragments are also discarded. All others are written.

**Parameters**

- **in_dat** (*file object*) – File handle in read mode to the 2D BED file containing Hi-C pairs.
- **out_filtered** (*file object*) – File handle in write mode the output filtered 2D BED file.
- **thr_uncut** (*int*) – Minimum number of restriction sites between reads to keep an intra-chromosomal +- pair.
- **thr_loop** (*int*) – Minimum number of restriction sites between reads to keep an intra-chromosomal -+ pair.
- **plot_events** (*bool*) – If True, a plot showing the proportion of each type of event will be shown after filtering.
- **fig_path** (*str*) – Path where the figure will be saved. If None, figure is displayed interactively.

- **prefix** (*str*) – If the library has a name, it will be shown on plots.

hicstuff.filter.**get_thresholds**(*in_dat*, *interactive=False*, *plot_events=False*, *fig_path=None*, *prefix=None*)

Guess distance threshold for event filtering

Analyse the events in the first million of Hi-C pairs in the library, plot the occurrences of each event type according to number of restriction fragments, and ask user interactively for the minimum threshold for uncuts and loops.

**Parameters**

- **in_dat** (*str*) – Path to the .pairs file containing Hi-C pairs.
- **interactive** (*bool*) – If True, plots are diplayed and thresholds are required interactively.
- **plot_events** (*bool*) – Whether to show the plot
- **fig_path** (*str*) – Path where the figure will be saved. If None, the figure will be diplayed interactively.
- **prefix** (*str*) – If the library has a name, it will be shown on plots.

**Returns** dictionary with keys "uncuts" and "loops" where the values are the corresponding thresholds entered by the user.

**Return type** dictionary

hicstuff.filter.**process_read_pair**(*line*)

Process and order read pairs in a .pairs record.

Takes a read pair (line) from a .pairs file as input, reorders the pair so that read 1 in intrachromosomal pairs always has the smallest genomic coordinate.

**Parameters** **line** (*str*) – Record from a pairs file with columns: readID, chr1, pos1, chr2, pos2, strand1, strand2, frag1, frag2

**Returns** Dictionary with reordered pair where each column from the line as an entry. The number of restriction sites separating reads in the pair and the type of event are also stored in the dictionary, under keys "nsites" and "type".

**Return type** dict

### Examples

```
>>> d = process_read_pair("readX\ta\t1\tb\t20\t-\t-\t1\t3")
>>> for u in sorted(d.items()):
...     print(u)
('chr1', 'a')
('chr2', 'b')
('frag1', 1)
('frag2', 3)
('nsites', 2)
('pos1', 1)
('pos2', 20)
('readID', 'readX')
('strand1', '-')
('strand2', '-')
('type', 'inter')
>>> d = process_read_pair('readY\ta\t20\ta\t10\t-\t+\t2\t1')
```

(continues on next page)

(continued from previous page)

```
>>> [d[x] for x in sorted(d.keys())]
['a', 'a', 1, 2, 1, 10, 20, 'readY', '+', '-', '+-']
```

### 2.1.6 hicstuff.hicstuff module

Common Hi-C functions

A bunch of handy functions for processing Hi-C data (mainly in the form of matrices):

- Normalizations
- Interpolations
- Filters
- Removing artifacts
- Quick sum-pooling (aka 'binning') in sparse and dense form
- Simple models with parameter estimation
- Computing best-matching 3D structures
- Various metrics in use among Hi-C people for eyecandy purposes (directional index, domainograms, etc.)

These functions are meant to be simple and relatively quick as-is implementations of procedures described in Hi-C papers.

hicstuff.hicstuff.**GC_partial**(*portion: str*)
Manually compute GC content percentage in a DNA string, taking ambiguous values into account (according to standard IUPAC notation).

**Parameters** **portion** (*str*) – DNA sequence on which GC content is computed.

**Returns** The percentage of GC in the input string.

**Return type** float

hicstuff.hicstuff.**GC_wide**(*genome: str*, *window=1000*)
Compute GC across a window of given length.

**Parameters**

- **genome** (*str*) – The genome on which GC content will be computed.
- **window** (*int*) – The window size in which GC content is measured.

**Note:** Biopython is required.

hicstuff.hicstuff.**asd**(*M1*, *M2*)
Compute a Fourier transform based distance between two matrices.

Inspired from Galiez et al., 2015 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4535829/)

**Parameters**

- **M1** (*array_like*) – The first (normalized) input matrix.
- **M2** (*array_like*) – The second (normalized) input matrix

**Returns** **asd** – The matrix distance

**Return type** numpy.float64

hicstuff.hicstuff.**bin_annotation**(*annotation=None*, *subsampling_factor=3*)
Perform binning on genome annotations such as contig information or bin positions.

hicstuff.hicstuff.**bin_bp_dense**(*M*, *positions*, *bin_len=10000*)
Perform binning with a fixed genomic length in base pairs on a dense matrix. Fragments will be binned such that their total length is closest to the specified input. If a contig list is specified, binning will be performed such that fragments never overlap two contigs. Fragments longer than bin size will not be split, which can result in larger bins. The last smaller bin of the chromosome will be merged with the previous one.

**Parameters**

- **M** (*2D numpy array of ints or floats*) – The Hi-C matrix to bin in dense format
- **positions** (*numpy array of int*) – List of 0-based basepair start positions of fragments bins
- **bin_len** (*int*) – Bin length in basepairs

**Returns**

- *2D numpy array of ints of floats* – Binned matrix
- *numpy array of ints* – List of binned fragments positions in basepair

hicstuff.hicstuff.**bin_bp_sparse**(*M*, *positions*, *bin_len=10000*)
Perform binning with a fixed genomic length in base pairs on a sparse matrix. Fragments will be binned such that their total length is closest to the specified input. Binning will be performed such that fragments never overlap two contigs. Fragments longer than bin size will not be split, which can result in larger bins. The last smaller bin of the chromosome will be merged with the previous one.

**Parameters**

- **M** (*sparse numpy matrix*) – Hi-C contact matrix in sparse format.
- **positions** (*numpy array of ints*) – Start positions of fragments in the matrix, in base pairs.
- **bin_len** (*int*) – Desired length of bins, in base pairs

**Returns**

- *sparse scipy sparse coo_matrix* – The binned sparse matrix in COO format.
- *list of ints* – The new bin start positions.

hicstuff.hicstuff.**bin_dense**(*M*, *subsampling_factor=3*)
Wraps the bin_sparse function to apply it on dense matrices. Bins are merged by groups of N to produce a lower resolution matrix.

**Parameters**

- **M** (*numpy.array*) – 2D array containing the Hi-C contact map
- **subsampling_factor** (*int*) – The number of bins to include in each group (subsample).

**Returns** **out_M** – The subsamples matrix, with a resolution lower than the input by a defined factor.

**Return type** numpy.array

hicstuff.hicstuff.**bin_exact_bp_dense**(*M*, *positions*, *bin_len=10000*)
Perform the kb-binning procedure with total bin lengths being exactly set to that of the specified input. Fragments overlapping two potential bins will be split and related contact counts will be divided according

**Parameters**

- **overlap proportions in each bin.** (*to*) –
- **M** (*2D numpy array of ints or floats*) – The Hi-C matrix to bin in dense format
- **positions** (*numpy array of int*) – List of basepair start positions of fragments bins
- **bin_len** (*int*) – Bin length in basepairs

**Returns**

- *2D numpy array of ints of floats* – Binned matrix
- *list* – List of binned fragments

hicstuff.hicstuff.**bin_matrix**(*M*, *subsampling_factor=3*)
Bin either sparse or dense matrices.

hicstuff.hicstuff.**bin_measurement**(*measurement=None*, *subsampling_factor=3*)
Perform binning on genome-wide measurements by summing each component in a window of variable size (subsampling_factor).

hicstuff.hicstuff.**bin_sparse**(*M*, *subsampling_factor=3*)
Bins a sparse matrix by combining bins into groups of user defined size. Binsize is independent of genomic coordinates. Remaining rows and cols are put into a smaller bin at the end.

**Parameters**

- **M** (*scipy.sparse.coo_matrix*) – The input Hi-C matrix in a sparse format.
- **subsampling_factor** (*int*) – The number of bins to include in each group (subsample).

**Returns** The subsamples matrix, with a resolution lower than the input by a defined factor.

**Return type** scipy.sparse.coo_matrix

hicstuff.hicstuff.**build_pyramid**(*M*, *subsampling_factor=3*)
Iterate over a given number of times on matrix M so as to compute smaller and smaller matrices with bin_dense.

hicstuff.hicstuff.**compartments**(*M*, *normalize=True*)
A/B compartment analysis

Perform a PCA-based A/B compartment analysis on a normalized, single chromosome contact map. The results are two vectors whose values (negative or positive) should presumably correlate with the presence of 'active' vs. 'inert' chromatin.

**Parameters**

- **M** (*array_like*) – The input, normalized contact map. Must be a single chromosome.
- **normalize** (*bool*) – Whether to normalize the matrix beforehand.

**Returns**

- **PC1** (*numpy.ndarray*) – A vector representing the first component.
- **PC2** (*numpy.ndarray*) – A vector representing the second component.

hicstuff.hicstuff.**compartments_sparse**(*M*, *normalize=True*)
A/B compartment analysis

Performs a detrending of the power law followed by a PCA-based A/B compartment analysis on a sparse, normalized, single chromosome contact map. The results are two vectors whose values (negative or positive) should presumably correlate with the presence of 'active' vs. 'inert' chromatin.

**Parameters**

- **M** (*array_like*) – The input, normalized contact map. Must be a single chromosome. Values are assumed to be only the upper triangle of a symmetrix matrix.
- **normalize** (*bool*) – Whether to normalize the matrix beforehand.
- **mask** (*array of bool*) – An optional boolean mask indicating which bins should be used

**Returns** **pr_comp** – An array containing the N first principal component

**Return type** numpy.ndarray

hicstuff.hicstuff.**contigs_to_positions**(*contigs*, *binning=10000*)

Build positions from contig labels

From a list of contig labels and a binning parameter, build a list of positions that's essentially a concatenation of linspaces with step equal to the binning.

**Parameters**

- **contigs** (*list or array_like*) – The list of contig labels, must be sorted.
- **binning** (*int, optional*) – The step for the list of positions. Default is 10000.

**Returns** **positions** – The piece-wise sorted list of positions

**Return type** numpy.ndarray

hicstuff.hicstuff.**corrcoef_sparse**(*A*, *B=None*)

Computes correlation coefficient on sparse matrices

**Parameters**

- **A** (*scipy.sparse.csr_matrix*) – The matrix on which to compute the correlation.
- **B** (*scipy.sparse.csr_matrix*) – An optional second matrix. If provided, the correlation between A and B is computed.

**Returns** The correlation matrix.

**Return type** scipy.sparse.csr_matrix

hicstuff.hicstuff.**despeckle_local**(*M*, *stds=2*, *width=2*)

Replace outstanding values (above stds standard deviations) in a matrix by the average of a surrounding window of desired width.

hicstuff.hicstuff.**despeckle_simple**(*B*, *th2=2*, *threads=1*)

Single-chromosome despeckling

Simple speckle removing function on a single chromomsome. It also works for multiple chromosomes but trends may be disrupted.

**Parameters**

- **B** (*scipy.sparse.csr*) – The input matrix to despeckle, in sparse (csr) format.
- **th2** (*float*) – The number of standard deviations above the mean beyond which despeckling should be performed
- **threads** (*int*) – The number of CPU processes on which the function can run in parallel.

**Returns** The despeckled matrix, in the same format it was given.

**Return type** array_like

hicstuff.hicstuff.**directional**(*M*, *window=None*, *circ=False*, *extrapolate=True*, *log=True*)
From a symmetrical matrix M of size n, return a vector d whose each component d[i] is a T-test of two samples represented by vectors of size window on either side of the i-th pixel on the diagonal. Edge elements may be extrapolated based on the vector size reduction, except in the case of circular genomes. If they aren't, d will be of size n - 2*(window-1) instead of n.

hicstuff.hicstuff.**distance_diagonal_law**(*matrix*, *positions=None*, *circular=False*)
Compute a distance law trend using the contact averages of equal distances. Specific positions can be supplied if needed.

hicstuff.hicstuff.**distance_law**(*size*, *prefactor=10000*, *gamma1=-0.5*, *gamma2=-1.5*, *inter=0.01*, *transition=None*)
Generate a theoretical matrix from the usual P(s) model with two exponents for short-scale and large-scale modes and a sigmoid to represent the transition inbetween.

**Parameters**

- **size** (*int*) – Size of the matrix (which will be of shape (size, size))
- **prefactor** (*float*) – Prefactor that's analogous to the coverage, by default 10000
- **gamma1** (*float, optional*) – Exponent for the short-scale mode, by default -0.5
- **gamma2** (*float, optional*) – Exponent for the large-scale mode, by default -1.5
- **inter** (*float, optional*) – Value for inter-chromosomal contacts that also represents the minimum value for intra-chromosomal contacts, by default 0.01
- **transition** (*int, optional*) – Coordinate of the transition between scale modes, by default 1/10 of the size

**Returns** A symmetrical Toeplitz matrix whose each diagonal represents a value of the P(s) model

**Return type** numpy.ndarray

hicstuff.hicstuff.**distance_law_from_mat**(*matrix*, *indices=None*, *log_bins=True*, *base=1.1*)
Compute distance law as a function of the genomic coordinate aka P(s). Bin length increases exponentially with distance if log_bins is True. Works on dense and sparse matrices. Less precise than the one from the pairs.

**Parameters**

- **matrix** (*numpy.array or scipy.sparse.coo_matrix*) – Hi-C contact map of the chromosome on which the distance law is calculated.
- **indices** (*None or numpy array*) – List of indices on which to compute the distance law. For example compartments or expressed genes.
- **log_bins** (*bool*) – Whether the distance law should be computed on exponentially larger bins.

**Returns**

- *numpy array of floats* – The start index of each bin.
- *numpy array of floats* – The distance law computed per bin on the diagonal

### Examples

```
>>> import numpy as np
>>> mat = np.array([[3, 2, 1], [2, 3, 2], [1, 2, 3]])
>>> idx, avg = distance_law_from_mat(mat, log_bins=False)
>>> idx
array([0, 1, 2])
>>> avg
array([3., 2., 1.])
```

hicstuff.hicstuff.**distance_to_contact**(*D*, *alpha=1*)
Compute contact matrix from input distance matrix. Distance values of zeroes are given the largest contact count otherwise inferred non-zero distance values.

hicstuff.hicstuff.**domainogram**(*M*, *window=None*, *circ=False*, *extrapolate=True*)
From a symmetrical matrix M of size n, return a vector d whose each component d[i] is the total sum of a square of 2*window+1 size centered on the i-th main diagonal element. Edge elements may be extrapolated based on the square size reduction (i.e. for window = 4, the first component will be equal to the first diagonal pixel multiplied by 81, the second one will be equal to the first 2x2 square on the diagonal multiplied by 81/4, etc.), except in the case of circular genomes. If they aren't, d will be of size n - 2*(window-1) instead of n.

hicstuff.hicstuff.**estimate_param_rippe**(*measurements*, *bins*, *init=None*, *circ=False*)
Perform least square optimization needed for the rippe_parameters function.

hicstuff.hicstuff.**flatten_positions_to_contigs**(*positions*)
Flattens and converts a positions array to a contigs array, if applicable.

hicstuff.hicstuff.**from_structure**(*structure*)
Return contact data from a 3D structure (in pdb format).

hicstuff.hicstuff.**get_good_bins**(*M*, *n_mad=2.0*, *s_min=None*, *s_max=None*, *symmetric=False*)
Filters out bins with outstanding sums using median and MAD of the log transformed distribution of bin sums. Only filters weak outlier bins unless *symmetric* is set to True.

**Parameters**

- **M** (*scipy sparse coo_matrix*) – Input sparse matrix representing the Hi-C contact map.
- **n_mad** (*float*) – Minimum number of median absolut deviations around median in the bin sums distribution at which bins will be filtered out.
- **s_min** (*float*) – Optional fixed threshold value for bin sum below which bins should be filtered out.
- **s_max** (*float*) – Optional fixed threshold value for bin sum above which bins should be filtered out.
- **symmetric** (*bool*) – If set to true, filters out outliers on both sides of the distribution. Otherwise, only filters out bins on the left side (weak bins).

**Returns** A 1D numpy array whose length is the number of bins in the matrix and values indicate if bins values are within the acceptable range (1) or considered outliers (0).

**Return type** numpy array of bool

hicstuff.hicstuff.**get_missing_bins**(*original*, *trimmed*)
Retrieve indices of a trimmed matrix with respect to the original matrix. Fairly fast but is only correct if diagonal values are different, which is always the case in practice.

hicstuff.hicstuff.**largest_connected_component**(*matrix*)
Compute the adjacency matrix of the largest connected component of the graph whose input matrix is adjacent.

hicstuff.hicstuff.**mad**(*M*, *axis=None*)
Computes median absolute deviation of matrix bins sums.

**Parameters**

- **M** (*scipy sparse coo_matrix*) – Sparse matrix in COO format.
- **axis** (*int*) – Compute MAD on rows if 0, on columns if 1 or on all pixels if None. If axis is None, MAD is computed only on nonzero pixels.

**Returns** MAD estimator of matrix bin sums

**Return type** float

hicstuff.hicstuff.**matrix_to_pdb**(*matrix*, *filename*, *contigs=None*, *annotations=None*, *indices=None*, *special_bins=None*, *alpha=1*)
Convert a matrix to a PDB file, shortcutting the intermediary generated structure.

hicstuff.hicstuff.**model_norm**(*matrix*, *positions=None*, *lengths=None*, *model='uniform'*, *circ=False*)

hicstuff.hicstuff.**noise**(*matrix*)
Just a quick function to make a matrix noisy using a standard Poisson distribution (contacts are treated as rare events).

hicstuff.hicstuff.**normalize_dense**(*M*, *norm='SCN'*, *order=1*, *iterations=40*)
Apply one of the many normalization types to input dense matrix. Will also apply any callable norms such as a user-made or a lambda function. NOTE: Legacy function for dense maps

**Parameters**

- **M** (*2D numpy array of floats*) –
- **norm** (*str*) – Normalization procedure to use. Can be one of "SCN", "mirnylib", "frag" or "global". Can also be a user- defined function.
- **order** (*int*) – Defines the type of vector norm to use. See numpy.linalg.norm for details.
- **iterations** (*int*) – Iterations parameter when using an iterative normalization procedure.

**Returns** Normalized dense matrix.

**Return type** 2D numpy array of floats

hicstuff.hicstuff.**normalize_sparse**(*M*, *norm='SCN'*, *iterations=40*, *n_mad=3.0*)
Applies a normalization type to a sparse matrix.

**Parameters**

- **M** (*scipy.sparse.csr_matrix of floats*) –
- **norm** (*str or callable*) – Normalization procedure to use. Can be one of "SCN" or "ICE". Can also be a user-defined function.
- **iterations** (*int*) – Iterations parameter when using an iterative normalization procedure.
- **n_mad** (*float*) – Maximum number of median absolute deviations of bin sums to allow for including bins in the normalization procedure. Bins more than *n_mad* mads below the median are excluded. Bins excluded from normalisation are set to 0.

**Returns** Normalized sparse matrix.

**Return type** scipy.sparse.csr_matrix of floats

hicstuff.hicstuff.**null_model**(*matrix*, *positions=None*, *lengths=None*, *model='uniform'*, *noisy=False*, *circ=False*, *sparsity=False*)
Attempt to compute a 'null model' of the matrix given a model to base itself on.

hicstuff.hicstuff.**pdb_to_structure**(*filename*)
Import a structure object from a PDB file.

hicstuff.hicstuff.**positions_to_contigs**(*positions*)
Label contigs according to relative positions

Given a list of positions, return an ordered list of labels reflecting where the positions array started over (and presumably a new contig began).

**Parameters** **positions** (*list or array_like*) – A piece-wise ordered list of integers representing positions

**Returns** **contig_labels** – The list of contig labels

**Return type** numpy.ndarray

hicstuff.hicstuff.**remove_inter**(*M*, *contigs*)
Remove interchromosomal contacts

Given a contact map and a list attributing each position to a given chromosome, set all contacts between each chromosome or contig to zero. Useful to perform calculations on intrachromosomal contacts only.

**Parameters**

- **M** (*array_like*) – The initial contact map
- **contigs** (*list or array_like*) – A 1D array whose value at index i reflect the contig label of the row i in the matrix M. The length of the array must be equal to the (identical) shape value of the matrix.

**Returns** **N** – The output contact map with no interchromosomal contacts

**Return type** numpy.ndarray

hicstuff.hicstuff.**remove_intra**(*M*, *contigs*, *mask*)
Remove intrachromosomal contacts

Given a contact map and a list attributing each position to a given chromosome, set all contacts within each chromosome or contig to zero. Useful to perform calculations on interchromosomal contacts only.

**Parameters**

- **M** (*array_like*) – The initial contact map
- **contigs** (*list or array_like*) – A 1D array whose value at index i reflect the contig label of the row i in the matrix M. The length of the array must be equal to the (identical) shape value of the matrix.

**Returns** **N** – The output contact map with no intrachromosomal contacts

**Return type** numpy.ndarray

hicstuff.hicstuff.**rippe_parameters**(*matrix*, *positions*, *lengths=None*, *init=None*, *circ=False*)
Estimate parameters from the model described in Rippe et al., 2001.

hicstuff.hicstuff.**scalogram**(*M*, *circ=False*, *max_range=False*)
Computes so-called 'scalograms' used to easily visualize contacts at different distance scales. Edge cases have been painstakingly taken care of.

**Parameters**

- **M1** (*array_like*) – The input contact map
- **circ** (*bool*) – Whether the contact map's reference genome is circular. Default is False.
- **max_range** (*bool or int*) – The maximum scale to be computed on the matrix. Default is False, which means the maximum possible range (len(M) // 2) will be taken.

**Returns** **N** – The output scalogram. Values that can't be computed due to edge issues, or being beyond max_range will be zero. In a non-circular matrix, this will result with a 'cone-shaped' contact map.

**Return type** array_like

hicstuff.hicstuff.**shortest_path_interpolation**(*matrix*, *alpha=1*, *strict=True*)
Perform interpolation on a matrix's data by using ShRec's shortest-path procedure backwards and forwards. This replaces zeroes with corresponding shortest-path based counts and may have the additional effect of 'blurring' the matrix somewhat. If strict is set to True, only zeroes are replaced this way.

Also known as Boost-Hi-C (https://www.ncbi.nlm.nih.gov/pubmed/30615061)

hicstuff.hicstuff.**simple_distance_diagonal_law**(*matrix*, *circular=False*)

hicstuff.hicstuff.**split_genome**(*genome*, *chunk_size=10000*)
Split genome into chunks of fixed size (save the last one).

hicstuff.hicstuff.**split_matrix**(*M*, *contigs*)
Split multiple chromosome matrix

Split a labeled matrix with multiple chromosomes into unlabeled single-chromosome matrices. Inter chromosomal contacts are discarded.

**Parameters**

- **M** (*array_like*) – The multiple chromosome matrix to be split
- **contigs** (*list or array_like*) – The list of contig labels

hicstuff.hicstuff.**subsample_contacts**(*M*, *n_contacts*)
Bootstrap sampling of contacts in a sparse Hi-C map.

**Parameters**

- **M** (*scipy.sparse.coo_matrix*) – The input Hi-C contact map in sparse format.
- **n_contacts** (*float*) – The number of contacts to be sampled if larger than one. The proportion of contacts to be sampled if between 0 and 1.

**Returns** A new matrix with a fraction of the original contacts.

**Return type** scipy.sparse.coo_matrix

hicstuff.hicstuff.**sum_mat_bins**(*mat*)
Compute the sum of matrices bins (i.e. rows or columns) using only the upper triangle, assuming symmetrical matrices.

**Parameters** **mat** (*scipy.sparse.csr_matrix*) – Contact map in sparse format, either in upper triangle or full matrix.

**Returns** 1D array of bin sums.

**Return type** numpy.array

hicstuff.hicstuff.**to_distance**(*matrix*, *alpha=1*)
Compute distance matrix from contact data by applying a negative power law (alpha) to its nonzero pixels, then interpolating on the zeroes using a shortest-path algorithm.

hicstuff.hicstuff.**to_pdb**(*structure*, *filename*, *contigs=None*, *annotations=None*, *indices=None*, *special_bins=None*)

From a structure (or matrix) generate the corresponding pdb file representing each chain as a contig/chromosome and filling the occupancy field with a custom annotation. If the matrix has been trimmed somewhat, remaining indices may be specified.

hicstuff.hicstuff.**to_structure**(*matrix*, *alpha=1*)

Compute best matching 3D genome structure from underlying input matrix using ShRec3D-derived method from Lesne et al., 2014.

Link: https://www.ncbi.nlm.nih.gov/pubmed/25240436

The method performs two steps: first compute distance matrix by treating contact data as an adjacency graph (of weights equal to a power law function of the data), then embed the resulting distance matrix into 3D space.

The alpha parameter influences the weighting of contacts: if alpha < 1 long-range interactions are prioritized; if alpha >> 1 short-range interactions have more weight wahen computing the distance matrix.

hicstuff.hicstuff.**trim_dense**(*M*, *n_mad=3*, *s_min=None*, *s_max=None*)

By default, return a matrix stripped of component vectors whose sparsity (i.e. total contact count on a single column or row) deviates more than specified number of standard deviations from the mean. Boolean variables s_min and s_max act as absolute fixed values which override such behaviour when specified.

**Parameters**

- **M** (*2D numpy array of floats*) – Dense Hi-C contact matrix
- **n_mad** (*int*) – Minimum number of standard deviation by which a the sum of contacts in a component vector must deviate from the mean to be trimmed.
- **s_min** (*float*) – Fixed minimum value below which the component vectors will be trimmed.
- **s_max** (*float*) – Fixed maximum value above which the component vectors will be trimmed.

**Returns** The input matrix, stripped of outlier component vectors.

**Return type** numpy 2D array of floats

hicstuff.hicstuff.**trim_sparse**(*M*, *n_mad=3*, *s_min=None*, *s_max=None*)

Apply the trimming procedure to a sparse matrix.

**Parameters**

- **M** (*scipy.sparse.coo_matrix*) – Sparse Hi-C contact map
- **n_mad** (*int*) – Minimum number of median absolute deviations by which a the sum of contacts in a component vector must deviate from the median to be trimmed.
- **s_min** (*float*) – Fixed minimum value below which the component vectors will be trimmed.
- **s_max** (*float*) – Fixed maximum value above which the component vectors will be trimmed.

**Returns** The input sparse matrix, stripped of outlier component vectors.

**Return type** scipy coo_matrix of floats

hicstuff.hicstuff.**trim_structure**(*struct*, *filtering='cube'*, *n=2*)

Remove outlier 'atoms' (aka bins) from a structure.

### 2.1.7 hicstuff.io module

hicstuff.io.**add_cool_column**(*clr*, *column*, *column_name*, *table_name='bins'*, *metadata={}*, *dtype=None*)
Adds a new column to a loaded Cooler store. If the column exists, it is replaced. This will affect the .cool file.

**Parameters**

- **clr** (*Cooler object*) – A Cooler store.
- **column** (*pandas Series*) – The column to add to the cooler.
- **column_name** (*str*) – The name of the column to add.
- **table_name** (*str*) – The name of the table to which the column should be added. Defaults to the "bins" table.
- **metadata** (*dict*) – A dictionary of metadata to associate with the new column.

hicstuff.io.**check_fasta_index**(*ref*, *mode='bowtie2'*)
Checks for the existence of a bowtie2 or bwa index based on the reference file name.

**Parameters**

- **ref** (*str*) – Path to the reference genome.
- **mode** (*str*) – The alignment software used to build the index. bowtie2 or bwa. If any other value is given, the function returns the reference path.

**Returns** **index** – The bowtie2 or bwa index basename. None if no index was found

**Return type** str

hicstuff.io.**check_is_fasta**(*in_file*)
Checks whether input file is in fasta format.

**Parameters** **in_file** (*str*) – Path to the input file.

**Returns** True if the input is in fasta format, False otherwise

**Return type** bool

hicstuff.io.**dade_to_graal**(*filename*, *output_matrix='abs_fragments_contacts_weighted.txt'*, *output_contigs='info_contigs.txt'*, *output_frags='abs_fragments_contacts_weighted.txt'*, *output_dir=None*)
Convert a matrix from DADE format (https://github.com/scovit/dade) to a graal-compatible format. Since DADE matrices contain both fragment and contact information all files are generated at the same time.

hicstuff.io.**flexible_hic_loader**(*mat*, *fragments_file=None*, *chroms_file=None*, *quiet=False*)
Wrapper function to load COO, bg2 or cool input and return the same output. COO formats requires fragments_file and chroms_file options. bg2 format can infer bin_size if fixed. When providing a bg2 matrix with uneven fragments length, one should provide fragments_file as well or empty bins will be truncated from the output.

**Parameters**

- **mat** (*str*) – Path to the matrix in graal, bedgraph2 or cool format.
- **fragments_file** (*str or None*) – Path to the file with fragments information (fragments_list.txt). Only required if the matrix is in graal format.
- **chroms_file** (*str or None*) – Path to the file with chromosome information (info_contigs.txt). Only required if the matrix is in graal format.
- **quiet** (*bool*) – If True, will silence warnings for empty outputs.

**Returns**

- **mat** (*scipy.sparse.coo_matrix*) – Sparse upper triangle Hi-C matrix.
- **frags** (*pandas.DataFrame or None*) – Table of fragment informations. None if information was not provided.
- **chroms** (*pandas.DataFrame or None*) – Table of chromosomes/contig information. None if information was not provided.

hicstuff.io.**flexible_hic_saver**(*mat*, *out_prefix*, *frags=None*, *chroms=None*, *hic_fmt='graal'*, *quiet=False*)

Saves objects to the desired Hi-C file format.

**Parameters**

- **mat** (*scipy.sparse.coo_matrix*) – Hi-C contact map.
- **out_prefix** (*str*) – Output path without extension (the extension is added based on hic_fmt).
- **frags** (*pandas.DataFrame or None*) – Table of fragments informations.
- **chroms** (*pandas.DataFrame or None*) – Table of chromosomes / contigs informations.
- **hic_fmt** (*str*) – Output format. Can be one of graal for graal-compatible COO format, bg2 for 2D bedgraph format, or cool for cooler compatible format.

hicstuff.io.**from_dade_matrix**(*filename*, *header=False*)

Load a DADE matrix

Load a numpy array from a DADE matrix file, optionally returning bin information from the header. Header data processing is delegated downstream.

**Parameters**

- **filename** (*str, file or pathlib.Path*) – The name of the file containing the DADE matrix.
- **header** (*bool*) – Whether to return as well information contained in the header. Default is False.

### Example

```
>>> import numpy as np
>>> import tempfile
>>> lines = [['RST', 'chr1~0', 'chr1~10', 'chr2~0', 'chr2~30'],
...          ['chr1~0', '5'],
...          ['chr1~10', '8', '3'],
...          ['chr2~0', '3', '5', '5'],
...          ['chr2~30', '5', '10', '11', '2']
...          ]
>>> formatted = ["\t".join(l) + "\n" for l in lines ]
>>> dade = tempfile.NamedTemporaryFile(mode='w')
>>> for fm in formatted:
...     dade.write(fm)
34
9
12
13
```

(continues on next page)

(continued from previous page)

```
18
>>> dade.flush()
>>> M, h = from_dade_matrix(dade.name, header=True)
>>> dade.close()
>>> print(M)
[[ 5.  8.  3.  5.]
 [ 8.  3.  5. 10.]
 [ 3.  5.  5. 11.]
 [ 5. 10. 11.  2.]]
```

```
>>> print(h)
['chr1~0', 'chr1~10', 'chr2~0', 'chr2~30']
```

See https://github.com/scovit/DADE for more details about Dade.

hicstuff.io.**gc_bins**(*genome_path*, *frags*)

Generate GC content annotation for bins using input genome.

**Parameters**

- **genome_path** (*str*) – Path the the genome file in FASTA format.
- **frags** (*pandas.DataFrame*) – Table containing bin segmentation of the genome. Required columns: chrom, start, end.

**Returns** GC content per bin, in the range [0.0, 1.0].

**Return type** numpy.ndarray of floats

hicstuff.io.**generate_temp_dir**(*path*)

Temporary directory generation

Generates a temporary file with a random name at the input path.

**Parameters** **path** (*str*) – The path at which the temporary directory will be created.

**Returns** The path of the newly created temporary directory.

**Return type** str

hicstuff.io.**get_hic_format**(*mat*)

Returns the format of the input Hi-C matrix

**Parameters** **mat** (*str*) – Path to the input Hi-C matrix

**Returns** Hi-C format string. One of graal, bg2, cool

**Return type** str

hicstuff.io.**get_pairs_header**(*pairs*)

Retrieves the header of a .pairs file and stores lines into a list.

**Parameters** **pairs** (*str or file object*) – Path to the pairs file.

**Returns** **header** – A list of header lines found, in the same order they appear in pairs.

**Return type** list of str

### Examples

```
>>> import os
>>> from tempfile import NamedTemporaryFile
>>> p = NamedTemporaryFile('w', delete=False)
>>> p.writelines(["## pairs format v1.0\n", "#sorted: chr1-chr2\n", "abcd\n"])
>>> p.close()
>>> h = get_pairs_header(p.name)
>>> for line in h:
...     print([line])
['## pairs format v1.0']
['#sorted: chr1-chr2']
>>> os.unlink(p.name)
```

hicstuff.io.**get_pos_cols**(*df*)

Get column names representing chromosome, start and end column from a dataframe. Allows flexible names.

**Parameters** **df** (*pd.DataFrame*) –

**Returns** Tuple containing the names of the chromosome, start and end columns in the input dataframe.

**Return type** tuple of str

### Examples

```
>>> import pandas as pd
>>> d = [1, 2, 3]
>>> df = pd.DataFrame(
...     {'Chromosome': d, 'Start': d, 'End': d, 'species': d}
... )
>>> get_pos_cols(df)
('Chromosome', 'Start', 'End')
>>> df = pd.DataFrame(
...     {'id': d, 'chr': d, 'start_bp': d, 'end_bp': d}
... )
>>> get_pos_cols(df)
('chr', 'start_bp', 'end_bp')
```

hicstuff.io.**getrandbits**(*k*) → x. Generates an int with k random bits.

hicstuff.io.**is_compressed**(*filename*)

Check compression status

Check if the input file is compressed from the first bytes.

**Parameters** **filename** (*str*) – The path to the input file

**Returns** True if the file is compressed, False otherwise.

**Return type** bool

hicstuff.io.**load_bedgraph2d**(*filename*, *bin_size=None*, *fragments_file=None*)

Loads matrix and fragment information from a 2D bedgraph file. Note this function assumes chromosomes are ordered in alphabetical. order

**Parameters**

- **filename** (*str*) – Path to the bedgraph2D file.
- **bin_size** (*int*) – The size of bins in the case of fixed bin size.

- **fragments_file** (*str*) – Path to a fragments file to explicitely provide fragments positions. If the matrix does not have fixed bin size, this prevents errors.

**Returns**

- **mat** (*scipy.sparse.coo_matrix*) – The Hi-C contact map as the upper triangle of a symetric matrix, in sparse format.
- **frags** (*pandas.DataFrame*) – The list of fragments/bin present in the matrix with their genomic positions.

hicstuff.io.**load_cool**(*cool*)

Reads a cool file into memory and parses it into graal style tables.

**Parameters** **cool** (*str*) – Path to the input .cool file.

**Returns**

- **mat** (*scipy coo_matrix*) – Hi-C contact map in COO format.
- **frags** (*pandas DataFrame*) – Table off bins matching the matrix. Corresponds to the content of the fragments_list.txt file.
- **chroms** (*pandas DataFrame*) – Table of chromosome informations.

hicstuff.io.**load_from_redis**(*key*)

Retrieve a dataset from redis

Retrieve a cached dataset that was stored in redis with the input key.

**Parameters** **key** (*str*) – The key of the dataset that was stored in redis.

**Returns** **M** – The retrieved dataset in array format.

**Return type** numpy.ndarray

hicstuff.io.**load_into_redis**(*filename*)

Load a file into redis

Load a matrix file and sotre it in memory with redis. Useful to pass around huge datasets from scripts to scripts and load them only once.

Inspired from https://gist.github.com/alexland/ce02d6ae5c8b63413843

**Parameters** **filename** (*str, file or pathlib.Path*) – The file of the matrix to load.

**Returns** **key** – The key of the dataset needed to retrieve it from redis.

**Return type** str

hicstuff.io.**load_pos_col**(*path*, *colnum*, *header=1*, *dtype=<class 'numpy.int64'>*)

Loads a single column of a TSV file with header into a numpy array.

**Parameters**

- **path** (*str*) – The path of the TSV file to load.
- **colnum** (*int*) – The 0-based index of the column to load.
- **header** (*int*) – Number of line to skip. By default the header is a single line.

**Returns** A 1D numpy array with the

**Return type** numpy.array

### Examples

```
>>> load_pos_col('test_data/mat_5kb.tsv', 0)[:10]
array([0, 0, 0, 0, 0, 0, 1, 1, 2, 2])
```

hicstuff.io.**load_sparse_matrix**(*mat_path*, *binning=1*, *dtype=<class 'numpy.float64'>*)

Load sparse matrix

Load a text file matrix into a sparse matrix object. The expected format is a 3 column file where columns are row_number, col_number, value. The first line consists of 3 values representing the total number of rows, columns and nonzero values.

**Parameters**

- **mat_path** (*file, str or pathlib.Path*) – The input matrix file in instagraal format.
- **binning** (*int or "auto"*) – The binning to perform. If "auto", binning will be automatically inferred so that the matrix size will not go beyond (10000, 10000) in shape. That can be changed by modifying the DEFAULT_MAX_MATRIX_SHAPE value. Default is 1, i.e. no binning is performed
- **dtype** (*type, optional*) – The type of data being loaded. Default is numpy.float64

**Returns** **sparse_mat** – The output (sparse) matrix in COOrdinate format.

**Return type** scipy.sparse.coo_matrix

### Examples

```
>>> S = load_sparse_matrix('test_data/mat_5kb.tsv', binning=1)
>>> S.data[:10]
array([84.,  2.,  3.,  2.,  1.,  1., 50.,  1., 66.,  1.])
>>> S.shape
(16, 16)
```

hicstuff.io.**read_compressed**(*filename*, *mode='r'*)

Read compressed file

Opens the file in read mode with appropriate decompression algorithm.

**Parameters** **filename** (*str*) – The path to the input file

**Returns** The handle to access the input file's content

**Return type** file-like object

hicstuff.io.**rename_genome**(*genome*, *output=None*, *ambiguous=True*)

Rename genome and slugify headers

Rename genomes according to a simple naming scheme; this is mainly done to avoid special character weirdness.

**Parameters**

- **genome** (*file, str or pathlib.Path*) – The input genome to be renamed and slugify.
- **output** (*file, str or pathlib.Path*) – The output genome to be written into. Default is <base>_renamed.fa, where <base> is genome_in without its extension.

- **ambiguous** (*bool*) – Whether to retain ambiguous non-N bases, otherwise rename them to Ns. Default is True.

hicstuff.io.**reorder_fasta**(*genome*, *output=None*, *threshold=100000*)
Reorder and trim a fasta file

Sort a fasta file by record lengths, optionally trimming the smallest ones.

**Parameters**

- **genome** (*str, file or pathlib.Path*) – The genome scaffold file (or file handle)
- **output** (*str, file or pathlib.Path*) – The output file to write to
- **threshold** (*int, optional*) – The size below which scaffolds are discarded, by default 100000

hicstuff.io.**save_bedgraph2d**(*mat*, *frags*, *out_path*)
Given a sparse matrix and a corresponding list of fragments, save a file in 2D bedgraph format.

**Parameters**

- **mat** (*scipy.sparse.coo_matrix*) – The sparse contact map.
- **frags** (*pandas.DataFrame*) – A structure containing the annotations for each matrix bin. Should correspond to the content of the fragments_list.txt file.

hicstuff.io.**save_cool**(*cool_out*, *mat*, *frags*, *metadata={}*)
Writes a .cool file from graal style tables.

**Parameters**

- **cool_out** (*str*) – Path to the output cool file.
- **mat** (*scipy coo_matrix*) – The Hi-C contact matrix in sparse COO format.
- **frags** (*pandas DataFrame*) – The graal style 'fragments_list' table.
- **metadata** (*dict*) – Potential metadata to associate with the cool file.

hicstuff.io.**save_sparse_matrix**(*s_mat*, *path*)
Save a sparse matrix

Saves a sparse matrix object into tsv format.

**Parameters**

- **s_mat** (*scipy.sparse.coo_matrix*) – The sparse matrix to save on disk
- **path** (*str*) – File path where the matrix will be stored

hicstuff.io.**sort_pairs**(*in_file*, *out_file*, *keys*, *tmp_dir=None*, *threads=1*, *buffer='2G'*)
Sort a pairs file in batches using UNIX sort.

**Parameters**

- **in_file** (*str*) – Path to the unsorted input file
- **out_file** (*str*) – Path to the sorted output file.
- **keys** (*list of str*) – list of columns to use as sort keys. Each column can be one of readID, chr1, pos1, chr2, pos2, frag1, frag2. Key priorities are according to the order in the list.
- **tmp_dir** (*str*) – Path to the directory where temporary files will be created. Defaults to current directory.
- **threads** (*int*) – Number of parallel sorting threads.

- **buffer** (*str*) – Buffer size used for sorting. Consists of a number and a unit.

hicstuff.io.**to_dade_matrix**(*M*, *annotations=''*, *filename=None*)

Returns a Dade matrix from input numpy matrix. Any annotations are added as header. If filename is provided and valid, said matrix is also saved as text.

### 2.1.8 hicstuff.iteralign module

Iterative alignment

Aligns iteratively reads from a 3C fastq file: reads are trimmed with a range-sweeping number of basepairs and each read set generated this way is mapped onto the reference genome. This may result in a small increase of properly mapped reads.

@author: Remi Montagne & cmdoret

hicstuff.iteralign.**filter_bamfile**(*temp_alignment*, *filtered_out*, *min_qual=30*)

Filter alignment BAM files

Reads all the reads in the input BAM alignment file. Write reads to the output file if they are aligned with a good quality, otherwise add their name in a set to stage them for the next round of alignment.

**Parameters**

- **temp_alignment** (*str*) – Path to the input temporary alignment.
- **outfile** (*str*) – Path to the output filtered temporary alignment.
- **min_qual** (*int*) – Minimum mapping quality required to keep a Hi-C pair.

**Returns** Contains the names reads that did not align.

**Return type** set

hicstuff.iteralign.**iterative_align**(*fq_in*, *tmp_dir*, *ref*, *n_cpu*, *bam_out*, *aligner='bowtie2'*, *min_len=20*, *min_qual=30*, *read_len=None*)

Iterative alignment

Aligns reads iteratively reads of fq_in with bowtie2, minimap2 or bwa. Reads are truncated to the 20 first nucleotides and unmapped reads are extended by 20 nucleotides and realigned on each iteration.

**Parameters**

- **fq_in** (*str*) – Path to input fastq file to align iteratively.
- **tmp_dir** (*str*) – Path where temporary files should be written.
- **ref** (*str*) – Path to the reference genome if Minimap2 is used for alignment. Path to the index genome if Bowtie2/bwa is used for alignment.
- **n_cpu** (*int*) – The number of CPUs to use for the iterative alignment.
- **bam_out** (*str*) – Path where the final alignment should be written in BAM format.
- **aligner** (*str*) – Choose between minimap2, bwa or bowtie2 for the alignment.
- **min_len** (*int*) – The initial length of the fragments to align.
- **min_qual** (*int*) – Minimum mapping quality required to keep Hi-C pairs.
- **read_len** (*int*) – Read length in the fasta file. If set to None, the length of the first read is used. Set this value to the longest read length in the file if you have different read lengths.

#### Examples

iterative_align(fq_in='example_for.fastq', ref='example_bt2_index', bam_out='example_for.bam', aligner="bowtie2") iterative_align(fq_in='example_for.fastq', ref='example_genome.fa', bam_out='example_for.bam', aligner="minimap2")

hicstuff.iteralign.**truncate_reads**(*tmp_dir*, *infile*, *unaligned_set*, *trunc_len*, *first_round*)
Trim read ends

Writes the n first nucleotids of each sequence in infile to an auxiliary. file in the temporary folder.

**Parameters**

- **tmp_dir** (*str*) – Path to the temporary folder.
- **infile** (*str*) – Path to the fastq file to truncate.
- **unaligned_set** (*set*) – Contains the names of all reads that did not map unambiguously in previous rounds.
- **trunc_len** (*int*) – The number of basepairs to keep in each truncated sequence.
- **first_round** (*bool*) – If this is the first round, truncate all reads without checking mapping.

**Returns** Path to the output fastq file containing truncated reads.

**Return type** str

### 2.1.9 hicstuff.main module

Simple Hi-C pipeline for generating and manipulating contact matrices.

**usage:** hicstuff [-hv] <command> [<args>...]

**options:**

| | |
|---|---|
| **-h, --help** | shows the help |
| **-v, --version** | shows the version |

**The subcommands are:** convert Convert Hi-C data between different formats. digest Digest genome into a list of fragments. cutsite Preprocess fastq files by digesting reads at religation site. distancelaw Analyse and plot distance law. filter Filters Hi-C pairs to exclude spurious events. iteralign Iteratively aligns reads to a reference genome. missview Preview missing Hi-C bins in based on the genome and read length. pipeline Hi-C pipeline to generate contact matrix from fastq files. rebin Bin the matrix and regenerate files accordingly. subsample Bootstrap subsampling of contacts from a Hi-C map. view Visualize a Hi-C matrix.

hicstuff.main.**main**()

### 2.1.10 hicstuff.version module

### 2.1.11 hicstuff.view module

Hi-C visualization

A lightweight library for quickly parsing, loading and viewing contact maps in instagraal or csv format.

hicstuff.view.**normalize**(*M*, *norm='SCN'*)
Attempt to normalize if hicstuff is found, does nothing otherwise.

hicstuff.view.**plot_matrix**(*array*, *ax=None*, *filename=None*, *vmin=0*, *vmax=None*, *title=None*, *dpi=500*, *cmap='Reds'*, *chrom_starts=None*)

A function that performs all the tedious matplotlib magic to draw a 2D array with as few parameters and as little whitespace as possible.

Adjusted from https://github.com/koszullab/metaTOR

**Parameters**

- **array** (*numpy.array*) – The input (dense) matrix that must be plotted.
- **ax** (*matplotlib.AxesSubplot*) – An optional axis on which to draw the plot. Useful for multipanel images.
- **filename** (*str*) – The filename where the image should be stored. If None, the image is shown interactively.
- **vmin** (*int*) – The minimum value on the colorscale.
- **vmax** (*int or None*) – The maximum value on the colorscale. If None, the 99th percentile is taken.
- **title** (*str*) – The string to display as figure title.
- **dpi** (*int*) – The DPI (i.e. resolution in dots per inch) of the output image.
- **cmap** (*str*) – The name of the matplotlib colormap to use when plotting the matrix.
- **chrom_starts** (*numpy.array*) – Array of bin positions where to draw chromosome starts as dotted lines.

hicstuff.view.**reorder_fasta**(*genome*, *output*, *threshold=100000*)

Reorder and trim a fasta file

Sort a fasta file by record lengths, optionally trimming the smallest ones.

**Parameters**

- **genome** (*str, file or pathlib.Path*) – The genome scaffold file (or file handle)
- **output** (*str, file or pathlib.Path*) – The output file to write to
- **threshold** (*int, optional*) – The size below which scaffolds are discarded, by default 100000

hicstuff.view.**scaffold_distribution**(*genome*, *threshold=100*, *plot=True*)

Visualize scaffold size distribution

Compute (and optionally display) scaffold size distribution for a genome in fasta format.

**Parameters**

- **genome** (*str, file or pathlib.Path*) – The genome scaffold file (or file handle)
- **threshold** (*int, optional*) – The size below which scaffolds are discarded, by default 1000000
- **plot** (*bool, optional*) – Whether to plot the results

**Returns** The list of scaffold sizes in decreasing order.

**Return type** list

hicstuff.view.**sparse_to_dense**(*M*, *remove_diag=True*)

Converts a sparse square matrix into a full dense matrix. Removes the diagonal by default.

**Parameters**

- **M** (*scipy.sparse.coo_matrix*) – A sparse representation of the matrix.
- **remove_diag** (*bool*) – Whether the diagonal

**Returns** The matrix in dense format.

**Return type** numpy.array

#### Example

```
>>> import numpy as np
>>> from scipy.sparse import coo_matrix
>>> row, col = np.array([1, 2, 3]), np.array([3, 2, 1])
>>> data = np.array([4, 5, 6])
>>> S = coo_matrix((data, (row, col)))
>>> M = sparse_to_dense(S, remove_diag=True)
>>> for u in M:
...     print(u)
[0 0 0 0]
[ 0  0  0 10]
[0 0 0 0]
[ 0 10  0  0]
```

### 2.1.12 Module contents

Stuff to do with Hi-C data

A simple library/pipeline to generate and handle simple Hi-C data.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## h

hicstuff, 53
hicstuff.commands, 13
hicstuff.digest, 24
hicstuff.distance_law, 26
hicstuff.filter, 31
hicstuff.hicstuff, 33
hicstuff.io, 43
hicstuff.iteralign, 50
hicstuff.main, 51
hicstuff.version, 51
hicstuff.view, 51

# Index


## A

AbstractCommand (*class in hicstuff.commands*), 13
add_cool_column() (*in module hicstuff.io*), 43
asd() (*in module hicstuff.hicstuff*), 33
attribute_fragments() (*in module hicstuff.digest*), 24
average_distance_law() (*in module hicstuff.distance_law*), 26

## B

bin_annotation() (*in module hicstuff.hicstuff*), 34
bin_bp_dense() (*in module hicstuff.hicstuff*), 34
bin_bp_sparse() (*in module hicstuff.hicstuff*), 34
bin_dense() (*in module hicstuff.hicstuff*), 34
bin_exact_bp_dense() (*in module hicstuff.hicstuff*), 34
bin_matrix() (*in module hicstuff.hicstuff*), 35
bin_measurement() (*in module hicstuff.hicstuff*), 35
bin_sparse() (*in module hicstuff.hicstuff*), 35
build_pyramid() (*in module hicstuff.hicstuff*), 35

## C

check_fasta_index() (*in module hicstuff.io*), 43
check_is_fasta() (*in module hicstuff.io*), 43
check_output_path() (*hicstuff.commands.AbstractCommand method*), 14
circular_distance_law() (*in module hicstuff.distance_law*), 26
compartments() (*in module hicstuff.hicstuff*), 35
compartments_sparse() (*in module hicstuff.hicstuff*), 35
contigs_to_positions() (*in module hicstuff.hicstuff*), 36
Convert (*class in hicstuff.commands*), 14
corrcoef_sparse() (*in module hicstuff.hicstuff*), 36
Cutsite (*class in hicstuff.commands*), 14

## D

dade_to_graal() (*in module hicstuff.io*), 43
data_transform() (*hicstuff.commands.View method*), 23
despeckle_local() (*in module hicstuff.hicstuff*), 36
despeckle_simple() (*in module hicstuff.hicstuff*), 36
Digest (*class in hicstuff.commands*), 15
directional() (*in module hicstuff.hicstuff*), 37
distance_diagonal_law() (*in module hicstuff.hicstuff*), 37
distance_law() (*in module hicstuff.hicstuff*), 37
distance_law_from_mat() (*in module hicstuff.hicstuff*), 37
distance_to_contact() (*in module hicstuff.hicstuff*), 38
Distancelaw (*class in hicstuff.commands*), 16
domainogram() (*in module hicstuff.hicstuff*), 38

## E

estimate_param_rippe() (*in module hicstuff.hicstuff*), 38
execute() (*hicstuff.commands.AbstractCommand method*), 14
execute() (*hicstuff.commands.Convert method*), 14
execute() (*hicstuff.commands.Cutsite method*), 15
execute() (*hicstuff.commands.Digest method*), 15
execute() (*hicstuff.commands.Distancelaw method*), 17
execute() (*hicstuff.commands.Filter method*), 17
execute() (*hicstuff.commands.Iteralign method*), 18
execute() (*hicstuff.commands.Missview method*), 18
execute() (*hicstuff.commands.Pipeline method*), 20
execute() (*hicstuff.commands.Rebin method*), 21
execute() (*hicstuff.commands.Scalogram method*), 21
execute() (*hicstuff.commands.Subsample method*), 22
execute() (*hicstuff.commands.View method*), 23
export_distance_law() (*in module hicstuff.distance_law*), 26

## F

Filter (*class in hicstuff.commands*), 17

filter_bamfile() (*in module hicstuff.iteralign*), 50
filter_events() (*in module hicstuff.filter*), 31
find_frag() (*in module hicstuff.digest*), 24
flatten_positions_to_contigs() (*in module hicstuff.hicstuff*), 38
flexible_hic_loader() (*in module hicstuff.io*), 43
flexible_hic_saver() (*in module hicstuff.io*), 44
frag_len() (*in module hicstuff.digest*), 24
from_dade_matrix() (*in module hicstuff.io*), 44
from_structure() (*in module hicstuff.hicstuff*), 38

## G

gc_bins() (*in module hicstuff.io*), 45
GC_partial() (*in module hicstuff.hicstuff*), 33
GC_wide() (*in module hicstuff.hicstuff*), 33
gen_enzyme_religation_regex() (*in module hicstuff.digest*), 24
generate_temp_dir() (*in module hicstuff.io*), 45
get_chr_segment_bins_index() (*in module hicstuff.distance_law*), 27
get_chr_segment_length() (*in module hicstuff.distance_law*), 27
get_distance_law() (*in module hicstuff.distance_law*), 27
get_good_bins() (*in module hicstuff.hicstuff*), 38
get_hic_format() (*in module hicstuff.io*), 45
get_missing_bins() (*in module hicstuff.hicstuff*), 38
get_names() (*in module hicstuff.distance_law*), 28
get_pairs_distance() (*in module hicstuff.distance_law*), 28
get_pairs_header() (*in module hicstuff.io*), 45
get_pos_cols() (*in module hicstuff.io*), 46
get_restriction_table() (*in module hicstuff.digest*), 25
get_thresholds() (*in module hicstuff.filter*), 32
get_ylim() (*in module hicstuff.distance_law*), 29
getrandbits() (*in module hicstuff.io*), 46

## H

hicstuff (*module*), 53
hicstuff.commands (*module*), 13
hicstuff.digest (*module*), 24
hicstuff.distance_law (*module*), 26
hicstuff.filter (*module*), 31
hicstuff.hicstuff (*module*), 33
hicstuff.io (*module*), 43
hicstuff.iteralign (*module*), 50
hicstuff.main (*module*), 51
hicstuff.version (*module*), 51
hicstuff.view (*module*), 51

## I

import_distance_law() (*in module hicstuff.distance_law*), 29
is_compressed() (*in module hicstuff.io*), 46
Iteralign (*class in hicstuff.commands*), 17
iterative_align() (*in module hicstuff.iteralign*), 50

## L

largest_connected_component() (*in module hicstuff.hicstuff*), 38
load_bedgraph2d() (*in module hicstuff.io*), 46
load_cool() (*in module hicstuff.io*), 47
load_from_redis() (*in module hicstuff.io*), 47
load_into_redis() (*in module hicstuff.io*), 47
load_pos_col() (*in module hicstuff.io*), 47
load_sparse_matrix() (*in module hicstuff.io*), 48
logbins_xs() (*in module hicstuff.distance_law*), 30

## M

mad() (*in module hicstuff.hicstuff*), 39
main() (*in module hicstuff.main*), 51
matrix_to_pdb() (*in module hicstuff.hicstuff*), 39
Missview (*class in hicstuff.commands*), 18
model_norm() (*in module hicstuff.hicstuff*), 39

## N

noise() (*in module hicstuff.hicstuff*), 39
normalize() (*in module hicstuff.view*), 51
normalize_dense() (*in module hicstuff.hicstuff*), 39
normalize_distance_law() (*in module hicstuff.distance_law*), 30
normalize_sparse() (*in module hicstuff.hicstuff*), 39
null_model() (*in module hicstuff.hicstuff*), 40

## P

parse_bin_str() (*in module hicstuff.commands*), 23
parse_ucsc() (*in module hicstuff.commands*), 23
pdb_to_structure() (*in module hicstuff.hicstuff*), 40
Pipeline (*class in hicstuff.commands*), 18
plot_matrix() (*in module hicstuff.view*), 51
plot_ps_slope() (*in module hicstuff.distance_law*), 30
positions_to_contigs() (*in module hicstuff.hicstuff*), 40
process_matrix() (*hicstuff.commands.View method*), 23
process_read_pair() (*in module hicstuff.filter*), 32

## R

read_compressed() (*in module hicstuff.io*), 48
Rebin (*class in hicstuff.commands*), 20
remove_inter() (*in module hicstuff.hicstuff*), 40

remove_intra() (*in module hicstuff.hicstuff*), 40
rename_genome() (*in module hicstuff.io*), 48
reorder_fasta() (*in module hicstuff.io*), 49
reorder_fasta() (*in module hicstuff.view*), 52
rippe_parameters() (*in module hicstuff.hicstuff*), 40

## S

save_bedgraph2d() (*in module hicstuff.io*), 49
save_cool() (*in module hicstuff.io*), 49
save_sparse_matrix() (*in module hicstuff.io*), 49
scaffold_distribution() (*in module hicstuff.view*), 52
Scalogram (*class in hicstuff.commands*), 21
scalogram() (*in module hicstuff.hicstuff*), 40
shortest_path_interpolation() (*in module hicstuff.hicstuff*), 41
simple_distance_diagonal_law() (*in module hicstuff.hicstuff*), 41
slope_distance_law() (*in module hicstuff.distance_law*), 31
sort_pairs() (*in module hicstuff.io*), 49
sparse_to_dense() (*in module hicstuff.view*), 52
split_genome() (*in module hicstuff.hicstuff*), 41
split_matrix() (*in module hicstuff.hicstuff*), 41
Subsample (*class in hicstuff.commands*), 21
subsample_contacts() (*in module hicstuff.hicstuff*), 41
sum_mat_bins() (*in module hicstuff.hicstuff*), 41

## T

to_dade_matrix() (*in module hicstuff.io*), 50
to_distance() (*in module hicstuff.hicstuff*), 41
to_pdb() (*in module hicstuff.hicstuff*), 41
to_structure() (*in module hicstuff.hicstuff*), 42
trim_dense() (*in module hicstuff.hicstuff*), 42
trim_sparse() (*in module hicstuff.hicstuff*), 42
trim_structure() (*in module hicstuff.hicstuff*), 42
truncate_reads() (*in module hicstuff.iteralign*), 51

## V

View (*class in hicstuff.commands*), 22

## W

write_frag_info() (*in module hicstuff.digest*), 25