Technical documentation for HiStrux package

Wiktor Wierzchowski, Adrian Zaręba

February 4, 2025

Contents

1 Introduction

This documentation includes detailed description of all modules and functions within *HiStrux* package as well as package configuration and environment setup. Package source code can be obtained through git repository.

2 Requirements and installation guide

This package is developed for the linux-64, osx-64, and osx-arm64 platforms and based on key dependencies available only through conda-forge (show in the table 1). Thus, to perform installation, authors recommend using package managers such as Conda, Miniforge or Mamba. For windows users using windows subsystem for linux can be advised.

Start by cloning the repository to your local machine:

```
$ git clone https://github.com/AdixPlaysGames/HiStrux.git
```

Enter created directory and create conda environment based on environment.yml file. Default name for environment is histrux_env. You can edit first line of environment.yml or rename it afterwards if you prefer.

```
$ cd HiStrux
$ micromamba create -f environment.yml
```

Verify weather environment was created and activate it.

```
$ micromamba env list
$ micromamba activate histrux_env
```

After this you can build package locally using:

```
$ pip install .
```

Package	**Version**
numpy	1.26.4
pandas	2.2.3
scipy	1.12.0
matplotlib	3.9.2
more-itertools	10.5.0
hoomd	4.8.2
cooler	0.10.2
h5py	3.11.0
gsd	3.3.2
pyvista	0.44.2
scikit-learn	1.6.0
seaborn	0.13.2
pandastable	0.13.1
ipykernel	6.29.5

Table 1: List of dependencies for*HiStrux* package.

3 Code documentation

The entire package is maintained in an appropriate format, ready to be installed using Mamba Forge. The internal modules, although separate, communicate with each other to ensure structured consistency. The structure of these modules includes:

- **eXtract Module**: Responsible for processing and analyzing scHi-C data with flexible parameter settings and visualization capabilities. It extracts derivative features of scH-C matrices for model training.
- **CycleSort Module**: Designed for implementing machine learning algorithms for classification and prediction based of scHi-C data.
- **reConstruct Module**: Focused on reconstructing chromatin structures throu iterative molecular simulations implemented using *HooMD-blue*.

Below, detailed break down of functinons present in those submodules is provided.

3.1 eXtract

3.1.1 process

Description:
Processes a single-cell Hi-C dataset to produce a contact matrix.

Parameters:

`cells` **(`pd.DataFrame`):** The input dataframe containing Hi-C data. Required columns include: `['cell_id', 'chromosome_1', 'start_1', 'chromosome_2', 'start_2', 'mapping_quality']`. Raises a `ValueError` if the input is not a pandas DataFrame or if required columns are missing.

`cell_id` **(Optional `str`, default=`None`):** Identifier for the specific cell to process. If `None`, the first `cell_id` in the dataframe will be used.

, default=`None`):] A list of tuples containing chromosome names and their lengths. Example: `[('chr1', 195471971), ('chr2', 182113224), ...]`.

`bin_size` **(Optional `int`, default=`1000000`):** The size of each bin in base pairs.

`selected_chromosomes` **(Optional `list[str]`, default=`None`):** A list of chromosome names to include in the analysis. If `None`, all chromosomes in `chromosome_lengths` are used.

`trans_interactions` **(Optional `bool`, default=`True`):** If `True`, include inter-chromosomal interactions. If `False`, only intra-chromosomal interactions are considered.

`mapping_quality_involved` **(Optional `bool`, default=`False`):** If `True`, the contact matrix will sum the mapping qualities for each bin. If `False`, it will count the number of interactions per bin.

`substring` **(Optional `int`, default=`2`):** Removes the last `substring` characters from chromosome names. For example, if `chromosome_1` is `chr1-P` and `substring=2`, it becomes `chr1`. Set to `None` to leave chromosome names unchanged.

Returns:

`np.ndarray`**:** A symmetric contact matrix (2D array) where rows and columns represent genomic bins.

Examples:

```
columns = [
    "chromosome_1", "start_1", "end_1",
    "chromosome_2", "start_2", "end_2",
    "cell_id", "read_id", "mapping_quality",
    "strand_1", "strand_2"
]
```

```
path = "path_to_file"

chromosome_lengths = [
    ('chr1', 195471971), ('chr2', 182113224),
    ...
]

# Process a single cell with a specific bin_size
cell_matrix = process(
    cell_df,
    cell_id='SCG0089_TCATGCCTCCCGTTAC-1',
    chromosome_lengths=chromosome_lengths,
    bin_size=500000,
    trans_interactions=False
)
print(cell_matrix)
```

3.1.2 calculate_cis_ab_comp

Description:
Calculates A/B compartments (`compartments_df`) for cis-contacts (intra-chromosomal) and then computes A-B contact statistics (`ab_stats_df`). This function utilizes helper routines like `compute_ab_compartments` and `compute_ab_stats` internally.

Parameters:

`contacts_df` **(`pd.DataFrame`):** Input DataFrame with contact information. Should include columns like `['chromosome_1', 'chromosome_2', 'start_1', 'end_1', 'start_2', 'end_2', 'cell_id', ...]`.

`bin_size` **(Optional `int`, default=`1000000`):** The bin size in base pairs.

`w` **(Optional `int`, default=`4`):** Parameter passed to the imputation function (if imputation is applied).

`p` **(Optional `float`, default=`0.85`):** Parameter passed to the imputation function (if imputation is applied).

`imputation_involved` **(Optional `bool`, default=`False`):** Whether to apply imputation on the contact matrix.

`plot` **(Optional `bool`, default=`False`):** If `True`, will display a plot for each chromosome to visualize the correlation matrix and PCA results.

Returns:

`tuple[pd.DataFrame, pd.DataFrame]`: A tuple containing:

- `compartments_df` (DataFrame with PC1 scores and compartment labels)
- `ab_stats_df` (DataFrame with A/B contact statistics)

Examples:

```
# Suppose we load the same data:
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')

# Trim unwanted chromosome suffix:
cell_df['chromosome_1'] = cell_df['chromosome_1'].str[:-2]
cell_df['chromosome_2'] = cell_df['chromosome_2'].str[:-2]

cell_df = cell_df[cell_df['cell_id'] == 'SCG0089_TCATGCCTCCCGTTAC-1']

ab_stats = calculate_cis_ab_comp(
```

```
    cell_df,
    bin_size=300000,
    w=4,
    p=0.85,
    imputation_involved=True,
    plot=False
)
print(ab_stats)

# Output could look like:
#   contact_type   count   fraction
# 0          AA     1779   0.402489
# 1          BB     2591   0.586199
# 2          AB       50   0.011312
```

3.1.3 compute_insulation_scores & compute_insulation_features

Description:
These two routines focus on computing and analyzing insulation scores from a given contact matrix.

- `compute_insulation_scores`: Calculates the insulation score for each bin (optionally applying smoothing), and can plot the results.
- `compute_insulation_features`: Analyzes local minima in those insulation scores, grouping them between local maxima, and computes various metrics (mean, sum, standard deviation, etc.).

Parameters (compute_insulation_scores):

`cell` (`np.ndarray`)**:** A 2D contact matrix.

`scale` (`int`)**:** The neighborhood size (number of bins around each bin) used in calculating insulation.

`apply_smoothing` **(Optional** `bool`**, default=**`True`**):** If `True`, performs local normalization (window-based); otherwise applies a global Z-score.

`plot` **(Optional** `bool`**, default=**`False`**):** If `True`, plots the insulation scores across bins.

Returns (compute_insulation_scores):

`np.ndarray`**:** A 1D array of insulation scores (either locally normalized or Z-scored).

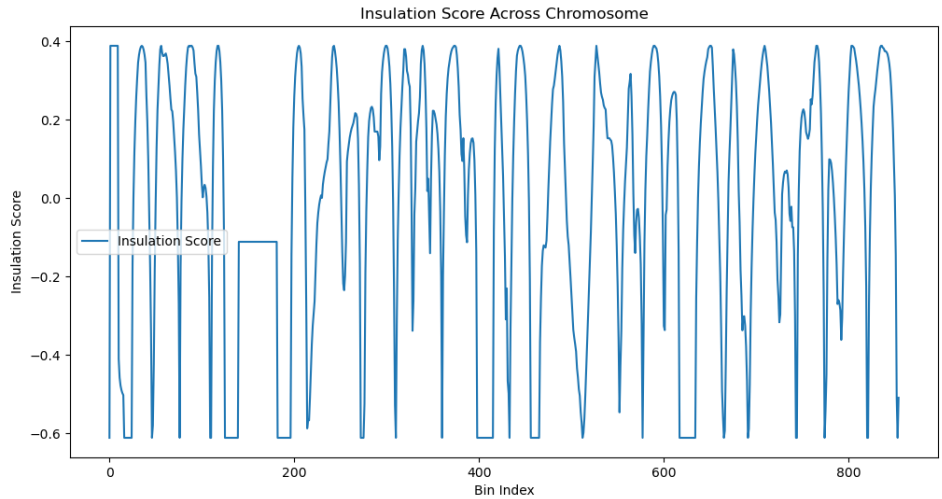
Parameters (compute_insulation_features):

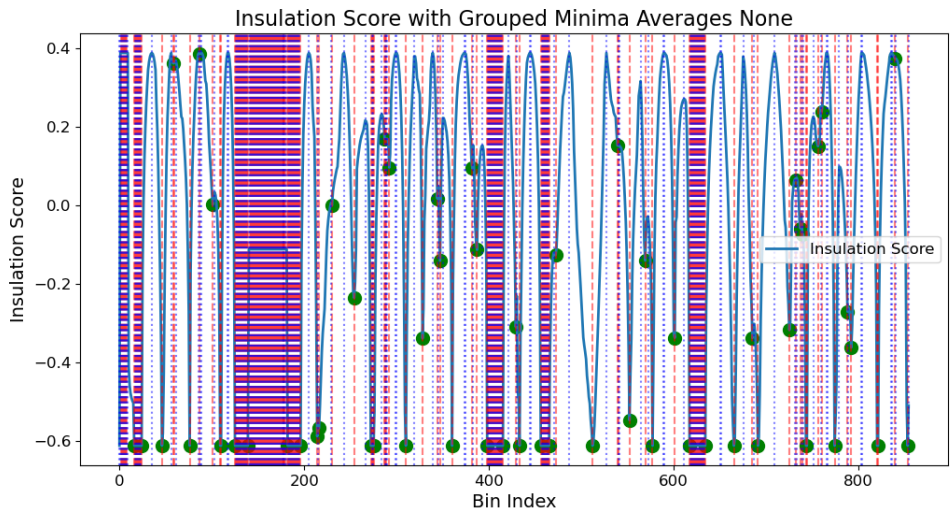
`ins_scores` (`np.ndarray`)**:** The 1D array of insulation scores, typically the output of `compute_insulation_scores`.

`plot` **(Optional** `bool`**, default=**`True`**):** If `True`, plots vertical lines for local minima/maxima and the group averages.

`chrom` **(Optional** `str`**, default=**`None`**):** Chromosome label for plotting/title purposes (if desired).

Returns (compute_insulation_features):





dict: A dictionary containing information about groups of local minima, group averages, mean/-sum/std of average minima values, etc.

Examples:

```
# Load data for a single chromosome:
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')
cell_matrix = process(
    cell_df,
    cell_id='SCG0089_TCATGCCTCCCGTTAC-1',
    chromosome_lengths=chromosome_lengths,
    bin_size=200000,
    selected_chromosomes=['chrX']
)

# Optional imputation (helper function)
cell_matrix = imputation(cell_matrix, w=5)

# Compute insulation scores
ins_scores = compute_insulation_scores(cell=cell_matrix, scale=15, plot=True)
```

```
print(ins_scores)

# Identify and analyze local minima
features = compute_insulation_features(ins_scores=ins_scores, plot=True)
print(features)

# A snippet of the output might look like:
# [ -0.6108  0.3891  0.3891 ... ]
# {
#   "groups": [...],
#   "group_averages": [...],
#   "mean_value": -0.3322,
#   "sum_value": -18.6061,
#   "std_deviation": 0.3254,
#   ...
# }
```

3.1.4 compute_mcm

Description:
Computes a multi-class metric (MCM) vector, reflecting the proportions of Hi-C contacts classified into three distance categories: `near`, `mid`, and `far`, based on user-defined distance thresholds in Mb.

Parameters:

`hic_matrix` **(`np.ndarray`):** A square Hi-C contact matrix (N x N). `hic_matrix[i, j]` is the contact count between bin i and j.

`bin_size` **(Optional `int`, default=`1000000`):** The size of each bin in base pairs (bp). Default is 1 Mb.

`near_threshold` **(Optional `float`, default=`2.0`):** Distance threshold (in Mb) below which contacts are considered `"near"`.

`mid_threshold` **(Optional `float`, default=`5.0`):** Distance threshold (in Mb) above which contacts are `"far"`. Contacts between `near_threshold` and `mid_threshold` are classified as `"mid"`.

Returns:

`dict`**:** A dictionary with three keys:

- `'mcm_near_ratio'`
- `'mcm_mid_ratio'`
- `'mcm_far_ratio'`

Each denotes the fraction of contacts in the respective distance category.

Examples:

```
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')
cell_matrix = process(cell_df, cell_id='SCG0089_TCATGCCTCCCGTTAC-1',
        bin_size=500_000)
print(compute_mcm(cell_matrix))
# Example output:
# {'mcm_near_ratio': 0.2067, 'mcm_mid_ratio': 0.1095, 'mcm_far_ratio': 0.6838}
```

3.1.5 compute_contact_scaling_exponent

Description:
Computes the contact scaling exponent in log-log scale (distance vs. average contact probability, *p(s)*) for a Hi-C contact matrix. It includes a basic Vanilla Coverage (VC) normalization, log-binning of distances, and an optional plot of the resulting curve with a best-fit line in log-log coordinates.

Parameters:

`contact_matrix` (`np.ndarray`): 2D scHi-C contact matrix to be analyzed.

`min_distance` **(Optional** `int`**, default=**`1`**):** Minimal distance (in bins) to consider.

`max_distance` **(Optional** `int`**, default=**`None`**):** Maximal distance (in bins) to include. If `None`, set to `N-1`.

`plot` **(Optional** `bool`**, default=**`False`**):** Whether to display a log-log plot of the distance vs. *p(s)* relationship.

`num_log_bins` **(Optional** `int`**, default=**`20`**):** Number of bins used in log-binning.

Returns:

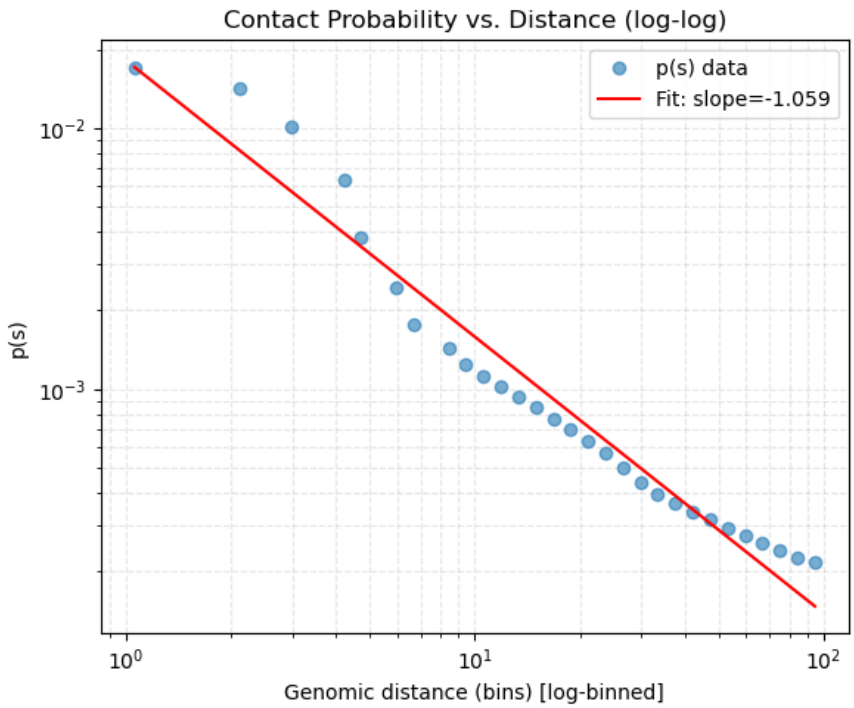


Figure 1: Insulation Score Smoothing

`dict`**:** A dictionary containing:

- `pofs_slope` (float)
- `pofs_intercept` (float)
- `pofs_r_value` (float)
- `pofs_p_value` (float)
- `pofs_std_err` (float)
- `pofs_distances` (np.ndarray) log-binned distance centers
- `p_of_s` (np.ndarray) average contact probabilities

Examples:

```
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')
cell_matrix = process(
    cell_df,
    cell_id='SCG0089_TCATGCCTCCCGTTAC-1',
    chromosome_lengths=chromosome_lengths,
    bin_size=500_000,
```

```
    trans_interactions=False
)

results = compute_contact_scaling_exponent(
    contact_matrix=cell_matrix,
    plot=True,
    num_log_bins=40,
    max_distance=100
)

# Example 'results["p_of_s"]' might look like:
# [0.0169, 0.0141, 0.0100, 0.0063, 0.0038, ... ]
# plus slope/intercept values from linear regression in log-log space.
```

3.1.6 compute_basic_metrics

Description:
Computes basic Hi-C interaction metrics for a given DataFrame. Internally uses the helper routines: `calculate_f_trans`, `calculate_mean_contact_length`, and `calculate_std_contact_length`.

Parameters:

`hic_df` (`pd.DataFrame`): A DataFrame containing Hi-C interaction data, with columns such as: `['chromosome_1', 'start_1', 'end_1', 'chromosome_2', 'start_2', 'end_2', 'cell_id', ...]`.

Returns:

`dict`: A dictionary with three keys:

- `'f_trans'`: The fraction of inter-chromosomal (`trans`) contacts.
- `'mean_contact_length'`: The mean contact length (averaged across both ends).
- `'std_contact_length'`: The standard deviation of contact lengths (averaged across both ends).

Examples:

```
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')
# Filter a single cell
cell_df = cell_df[cell_df['cell_id'] == 'SCG0088_TTGTGTGCACGGTACT-1']

print(compute_basic_metrics(cell_df))
# Example output:
# {'f_trans': 0.1623,
    'mean_contact_length': 102.1490, 'std_contact_length': 51.1887}
```

3.1.7 compute_tad_features

Description:
An orchestrator function that detects TADs (Topologically Associating Domains) in cis-contacts and computes various TAD-based features across chromosomes. Internally, it uses multiple helper functions (e.g., `calculate_cis_tads`, `compute_directionality_index`, `detect_tad_boundaries`, `build_tad_df`, `plot_tads_for_chrom`, etc.), but the primary entry point is `compute_tad_features`.

Parameters:

`contacts_df` (`pd.DataFrame`): Input DataFrame containing contact information (e.g. `chromosome_1, start_1, chromosome_2, start_2`).

`bin_size` (Optional `int`, default=`1000000`): The bin size in base pairs (default 1 Mb).

`w` (Optional `int`, default=`10`): Window size for directionality index (and possibly imputation).

`p` **(Optional** `float`**, default=**`0.85`**):** Probability parameter for random walk with restart in imputation (if used).

`imputation_involved` **(Optional** `bool`**, default=**`False`**):** Whether to apply an imputation step on the contact matrix.

`boundary_threshold` **(Optional** `float`**, default=**`0.3`**):** Threshold (in terms of directionality index sign changes) for calling TAD boundaries.

`out_prefix` **(Optional** `str`**, default=**`None`**):** If not `None`, saves plots with this prefix for each chromosome. Otherwise, no file is saved.

`show_plot` **(Optional** `bool`**, default=**`False`**):** Whether to display TAD plots via `plt.show()`.

Returns:

`dict`**:** A dictionary with keys:

- `"tad_n_tads_mean"`: Geometric mean of the number of TADs per chromosome.
- `"tad_mean_bin_size"`: Geometric mean of mean TAD sizes (in bins).
- `"tad_density_mean"`: Geometric mean of the TAD density (TADs per bin).

Examples:

```
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')
cell_df = cell_df[cell_df['cell_id'] == 'SCG0089_TCATGCCTCCCGTTAC-1']
cell_df['chromosome_1'] = cell_df['chromosome_1'].str[:-2]
cell_df['chromosome_2'] = cell_df['chromosome_2'].str[:-2]

tad_metrics = compute_tad_features(
    cell_df,
    bin_size=600000,
    w=3,
    p=0.85,
    imputation_involved=True,
    boundary_threshold=0.05,
    show_plot=False
)

print(tad_metrics)
# Example output:
# {
#   'tad_n_tads_mean': 4.518981561277916,
#   'tad_mean_bin_size': 46.73657879942423,
#   'tad_density_mean': 0.021396516940010157
# }
```

3.1.8 visualize

Description:
Simple matrix visualization function that takes a 2D contact matrix (`np.ndarray`) and plots it as a heatmap.

Parameters:

`matrix` **(**`np.ndarray`**):** The 2D contact matrix to be visualized. Must be a square array.

`title` **(Optional** `str`**, default=**`'scHi-C'`**):** Plot title.

`xlabel` **(Optional** `str`**, default=**`'Genome position 1'`**):** Label for the x-axis.

`ylabel` **(Optional** `str`**, default=**`'Genome position 2'`**):** Label for the y-axis.

`return_plot` **(Optional** `bool`**, default=**`False`**):** Currently not used to return a figure, but could be extended if needed.

Returns:

`None`**:** Displays the heatmap of the given contact matrix.

Examples:

```
from eXtract.visualization import visualize
import numpy as np

# Suppose we have a 10x10 matrix:
test_matrix = np.random.randint(0, 100, size=(10, 10))
visualize(test_matrix, title='Random␣Contact␣Matrix')
```

3.1.9 eXtract

Description:
A master function that processes Hi-C data from a single cell and computes multiple metrics in one go. Internally, it uses:

- `process` (to generate a contact matrix),
- `calculate_cis_ab_comp` (compartments),
- `compute_mcm`,
- `compute_contact_scaling_exponent` (*p(s)*),
- `compute_basic_metrics`,
- `compute_tad_features`,
- an *insulation analysis* via `compute_ins_features_for_each_chr`.

The function can optionally display a `tkinter` table with results, or return a vector of values.
Parameters:

`cell_dataframe` **(**`pd.DataFrame`**):** A DataFrame containing Hi-C data (`chromosome_1, start_1, end_1, chromosome_2, start_2, end_2, cell_id`, etc.).

`cell_id` **(Optional** `str`**, default=**`None`**):** Identifier for which cell to extract. If `None`, the first `cell_id` in `cell_dataframe` is used.

, default=`None`):] List of (chromosome name, chromosome length). If not provided, defaults may apply inside `process`.

`bin_size` **(Optional** `int`**, default=**`500000`**):** Bin size (in bp) for `process`.

`selected_chromosomes` **(Optional** `list[str]`**, default=**`None`**):** List of chromosomes to be processed. If `None`, all are used.

`trans_interactions` **(Optional** `bool`**, default=**`True`**):** If `True`, `process` includes trans contacts in the returned matrix.

`mapping_quality_involved` **(Optional** `bool`**, default=**`False`**):** If `True`, uses mapping qualities in the contact matrix weighting.

`substring` **(Optional, default=**`2`**):** Number of trailing characters to remove from chromosome names. Set `None` to skip removal.

`compartments_w` **(Optional** `int`**, default=**`4`**):** Window size for compartment imputation. Passed to `calculate_cis_ab_comp`.

`compartments_p` **(Optional `float`, default=`0.85`):** Probability parameter for compartment imputation. Passed to `calculate_cis_ab_comp`.

`compartments_bin_size` **(Optional `int`, default=`400000`):** Bin size (in bp) for compartment calculation.

`ins_bin_size` **(Optional `int`, default=`400000`):** Bin size for the insulation score computation.

`scale` **(Optional `int`, default=`15`):** Scale/neighborhood size for the insulation function.

`ins_p` **(Optional `float`, default=`0.85`):** Probability parameter for insulation matrix imputation.

`ins_w` **(Optional `int`, default=`3`):** Window size used in the insulation function.

`near_threshold` **(Optional `float`, default=`2.0`):** Distance threshold (in Mb) below which contacts count as *near*.

`mid_threshold` **(Optional `float`, default=`6.0`):** Distance threshold (in Mb) above which contacts count as *far* (the mid range is in between).

`min_distance` **(Optional `int`, default=`1`):** Minimal distance in bins for `compute_contact_scaling_exponent`.

`max_distance` **(Optional `int`, default=`100`):** Maximal distance in bins for `compute_contact_scaling_exponent`.

`pofs_bin_size` **(Optional `int`, default=`500000`):** Bin size for the *p(s)* contact matrix (cis-only).

`num_log_bins` **(Optional `int`, default=`40`):** Number of log-bins for *p(s)* analysis.

`tad_bin_size` **(Optional `int`, default=`300000`):** Bin size (in bp) used by `compute_tad_features`.

`tad_w` **(Optional `int`, default=`3`):** Window for directionality index in TAD detection.

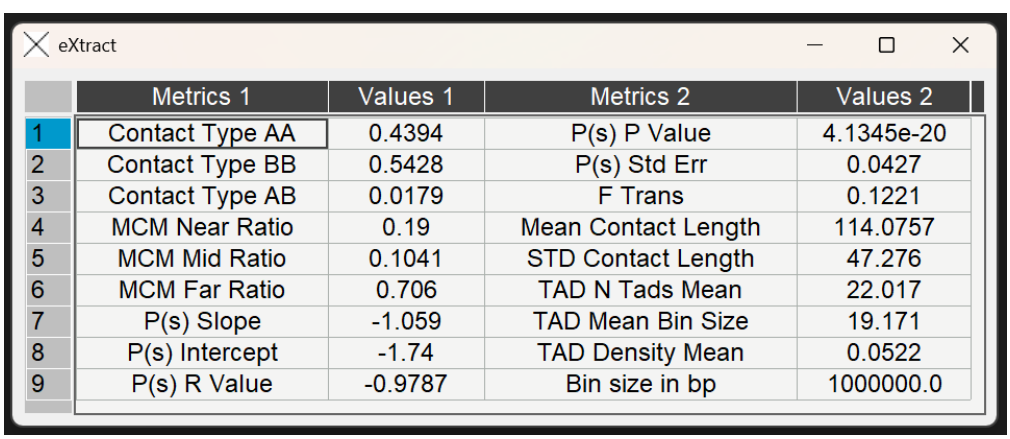
`tad_boundry_threshold` **(Optional `float`, default=`0.05`):** Threshold for calling TAD boundaries from the directionality index.

`out_prefix` **(Optional `str`, default=`None`):** If not `None`, `compute_tad_features` can save TAD plots with this prefix.

`vectorize` **(Optional `bool`, default=`False`):**

- If `False`, returns (and optionally displays) a `pd.DataFrame` with all the metrics.
- If `True`, returns two lists: one with the metric names, and one with the numeric values (a feature vector).

Returns:



eXtract

	Metrics 1	Values 1	Metrics 2	Values 2
1	Contact Type AA	0.4394	P(s) P Value	4.1345e-20
2	Contact Type BB	0.5428	P(s) Std Err	0.0427
3	Contact Type AB	0.0179	F Trans	0.1221
4	MCM Near Ratio	0.19	Mean Contact Length	114.0757
5	MCM Mid Ratio	0.1041	STD Contact Length	47.276
6	MCM Far Ratio	0.706	TAD N Tads Mean	22.017
7	P(s) Slope	-1.059	TAD Mean Bin Size	19.171
8	P(s) Intercept	-1.74	TAD Density Mean	0.0522
9	P(s) R Value	-0.9787	Bin size in bp	1000000.0

Figure 2: Features eXtract table.

`pd.DataFrame` **or** `(list, list)`:

- If `vectorize=False`, returns a `DataFrame` (and also displays it in a *tkinter* window) containing the computed metrics: `[AA/BB/AB compartments, MCM near/mid/far ratios, P(s) slope/intercept, ... , TAD features, etc.]`.
- If `vectorize=True`, returns a tuple of two lists: `(metricNames, metricValues)` suitable for vectorized representations of the metrics.

Examples:

```
# Usage example:
cell_id = 'SCG0089_TCATGCCTCCCGTTAC-1'
cell_df = pd.read_csv(path, sep="\t", names=columns, comment='#')

# Example 1: Return a DataFrame (and display it via Tkinter) with all metrics:
extracted_df = eXtract(
    cell_dataframe=cell_df,
    cell_id=cell_id,
    bin_size=1_000_000,
    vectorize=False
)
print(extracted_df)

# Example 2: Return a feature vector (two lists: names, values):
names, values = eXtract(
    cell_dataframe=cell_df,
    cell_id=cell_id,
    bin_size=1_000_000,
    vectorize=True
)
print(names)
print(values)
```

3.2 CycleSort

3.2.1 CycleSort (Classification Project)

Description:
Although `CycleSort` (the code snippet below) is not directly part of the core `HiStrux` package, it represents an example of a classification model pipeline that leverages data extracted by `eXtract` and can be used in conjunction with `reConstruct`. In essence, this classification module demonstrates how one can employ the features from `eXtract` to build and train models for cell-stage or cell-cycle classification (or similar analyses), thereby showcasing the broader applicability and extensibility of the `eXtract` outputs in various machine learning pipelines.

Key Steps:

1. **Data Aggregation:** Reads multiple `.csv` files from a folder, tagging them with a putative cell-cycle stage (`S`, `G1`, `G2M`) based on filename.
2. **Feature Preparation:** Merges the data and drops duplicates by `cell_id`.
3. **Data Encoding:** Encodes the stages into numeric vectors, suitable for regression-based or vector-distance-based classification.
4. **Scaling and Splitting:** Scales features to $[0, 1]$ using `MinMaxScaler`, then splits into training and test sets.
5. **Neural Network:** Uses a small `Keras` sequential model (two hidden layers) to learn a mapping of features to the 3D vectors denoting the stages.
6. **Post-Training Mapping:** Maps the network outputs to the nearest known class vectors and assesses accuracy using a confusion matrix.

7. **3D Visualization:** Employs `plotly` for an interactive 3D scatter plot of both train and test data.

Returns:

`accuracy_score`**:** The final classification accuracy of the model.

`3D interactive figure`**:** An on-screen 3D scatter plot of embedded features with class color coding.

`confusion matrix`**:** Printed confusion matrix of true vs. predicted classes.

Example Code Snippet:

```
import pandas as pd
import os
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Dense
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go

# Load multiple CSVs from a folder, label by cell-cycle stage:
folder_path = 'patski'
patski_df = pd.DataFrame()
for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):
        new_df = pd.read_csv(os.path.join(folder_path, file_name), sep=';')
        # Stage inference by filename:
        if 'S_' in file_name:
            new_df['Stage'] = 'S'
        elif 'G1_' in file_name:
            new_df['Stage'] = 'G1'
        elif 'G2M_' in file_name:
            new_df['Stage'] = 'G2M'
        else:
            new_df['Stage'] = 'Unknown'
        patski_df = pd.concat([patski_df, new_df], ignore_index=True)

patski_df = patski_df.drop_duplicates('cell_id')
# Assign numeric vectors to each Stage...
# [... etc. ...]
```

```
# Build & train a neural network
# Evaluate accuracy, produce a 3D scatter plot, confusion matrix, etc.
```

3.3 reConstruct - data selection

3.3.1 load_cells_names

Description:
Reads required number of cell names from scool file taking first cells. If user knows the names of cells he wants to work with this function can be used as a check whether those cells are actually in the file.

Parameters:

`population_dir` **(`str`):** Scool file directory.

`num` **(Optional `int`, default=`None`):** Number of cell names to be read.

`cells_names` **(Optional, `list[string]`, default=`None`):** List of cell names to be checked for presence.

Returns:

`list`**:** A list of cell names.

Examples:

```
>>> load_cells_names('/data/my_population.scool', 3)
['Cell1', 'Cell2', 'Cell3']

>>> load_cells_names('/data/my_population.scool', ['Cell1', 'Cell2', 'Cell4'])
['Cell1', 'Cell2']
```

3.3.2 load_data

Description:
Extracts scHiC contact matrix and bins series describing this matrix. Data is loaded from cool file data source. If user works with composed population data within scool file the cell selection need to follow format: $scool\_file\_directory/scool\_file\_name :: cell\_name$. It picks only specified chromosomes. Functions *remove_diag_plus* and *normalize_hic* are included in it by default, but can be turned of with a parameter.

Parameters:

`cell_dir` **(`str`):** Scool file directory with cell name.

`chroms_list` **(`list[str]`):** List of chromosome names to loaded.

`do_not_clean` **(Optional `bool`, default=`False`):** True/False value controling whether normalization and main diagonal removal takes place.

`normalization_percentile` **(Optional `int`, default=`90`):** Percentile of weakest contacts to be removed from contact matrices in.

Returns:

`tuple[numpy.ndarray, pandas.Series]` Touple of n-dim numpy array with scHiC contact matrix and pandas series of bins describing contacts chromosome, start and end of chromatine fragment.

Examples:

```
>>> load_data('/data/my_population.scool::my_cell', ['chr10', 'chr11',
'chr12', 'chr13', 'chr14', 'chr15'])
    (array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]),
        chrom      start        end
    0    chr10          0    1000000
    1    chr10    1000000    2000000
    2    chr10    2000000    3000000
    3    chr10    3000000    4000000
    4    chr10    4000000    5000000
    ..     ...        ...        ...
    721  chr15   99000000  100000000
    722  chr15  100000000  101000000
    723  chr15  101000000  102000000
    724  chr15  102000000  103000000
    725  chr15  103000000  103494974
```

3.3.3 remove_diag_plus

Description:
Sets main diagonal of matrix, and together with diagonals one above and below it to zero. This function is used as part of preprocessing to remove bins contacts on diagonal and those of neighboring bins which are not useful in chromatin reconstruction. It is used by default in load_data function.

Parameters:

`matrix` (`np.ndarray`)**:** Numpy square ndarray.

Returns:

`np.ndarray` Modified matrix.

Examples:

```
>>> matrix = np.ndarray((4,4))
    >>> matrix[:] = [[1, 2, 3, 4],
                     [5, 6, 7, 8],
                     [9, 10, 11, 12],
                     [13, 14, 15, 16]]
    >>> remove_diag_plus(matrix)
    >>> print(matrix)
    array([[0, 0, 3, 4],
           [0, 0, 0, 8],
           [9, 0, 0, 0],
           [13, 14, 0, 0]])
```

3.3.4 normalize_hic

Description:
Applies natural logarithm over matrix values and sets p-th percentile of lowest values to 0. This function is part of data preprocessing and is used by default in load_data function.

Parameters:

`hic` (`np.ndarray`)**:** Numpy square ndarray.

Returns:

`np.ndarray` Modified matrix.

Example:

```
>>> matrix = np.ndarray((4,4))
    >>> matrix[:] = [[0, 0, 3, 4], [0, 0, 0, 8], [9, 0, 0, 0], [13, 14, 0, 0]]
    >>> normalize_hic(matrix)
    >>> print(matrix)
    array([[0, 0, 0, 0],
           [0, 0, 0, 0],
           [0, 0, 0, 0],
           [13, 14, 0, 0]])
```

3.3.5 filter_poor_cells

Description:
This function filters list of cells names and returns only cells which contact matrices have required minimal number of contacts as well as minimal and maximal ratio of long range contacts. Where long range contacts are defined as one's located beyond main_width central diagonals of matrix.

Parameters:

`population_dir` (`str`)**:** Scool file directory containing cells data.

`cells_names` (`list[str]`)**:** List of cells names to be filtered.

`chroms_list` (`list[str]`)**:** List of chromosome names to be considered.

`min_contacts` **(Optional** `int`**, default=**`4000`**):** Requires number of contacts to be higher than that.

`min_ratio` **(Optional** `int`**, default=**`0.15`**):** Requires ratio of long range contacts to be higher than that.

`max_ratio` **(Optional** `int`**, default=**`0.4`**):** Requires ratio of long range contacts to be lower than that.

`main_width` **(Optional** `int`**, default=**`50`**):** Defines long range contacts. If contacts is located on diagonal further than 50 diagonals up or down main diagonal of matrix it is constidered long range.

Returns:

`numpy.array` Array of cells names fulfilling filter requirements.

Example:

```
>>> filter_poor_cells('/data/my_population.scool', ['Cell1', 'Cell2', 'Cell3'],
['chr10', 'chr11', 'chr12'], min_contacts=4500, min_ratio=0.20,
max_ratio=0.50, main_width=60)
    array(['Cell1'], dtype='<U38')
```

3.3.6 sample_series

Description:
Samples desired number of cells from provided cells population. Keep input ratios of labels. Function returns three arrays with sampled cells names, labels and prediction values, all crucial for further steps.

Parameters:

`cells_names` (`list[str]`)**:** List of cells names to be filtered.

`labels` (`list[int]`): List of interphase labels given to the cells.

`predictions` (`list[np.ndarray]`): List of numpy ndarrays describing prediction.

`series_size` (`int`): Number of cells to be sampled.

Returns:

`numpy.array` Array of names of cells sampled in a series.

`numpy.array` Array of labels of cells sampled in a series.

`numpy.array` Array of prediction values of cells sampled in a series.

Examples:

```
>>> print(filtered_population_names)
    ['Diploid_10_ACTGAGCG_AAGGCTAT_R1fastqgz'
    'Diploid_10_ACTGAGCG_CCTAGAGT_R1fastqgz'
    'Diploid_10_ACTGAGCG_CTATTAAG_R1fastqgz'
    'Diploid_10_ACTGAGCG_GAGCCTTA_R1fastqgz'
    'Diploid_10_ACTGAGCG_GCGTAAGA_R1fastqgz'
    'Diploid_10_ACTGAGCG_TCGACTAG_R1fastqgz'
    'Diploid_10_ATGCGCAG_AAGGCTAT_R1fastqgz'
    'Diploid_10_ATGCGCAG_CCTAGAGT_R1fastqgz'
    'Diploid_10_ATGCGCAG_CTATTAAG_R1fastqgz'
    'Diploid_10_ATGCGCAG_GCGTAAGA_R1fastqgz']
    >>> print(population_labels)
    [0 2 2 2 2 0 2 0 0 1]
    >>> print(population_predictions)
    [[0.50630042 0.2317334  0.26196618]
    [0.35175224 0.23726902 0.41097873]
    [0.35609426 0.02918836 0.61471738]
    [0.15329386 0.26029309 0.58641306]
    [0.31415687 0.17877836 0.50706477]
    [0.53080315 0.21029382 0.25890303]
    [0.32800212 0.03946776 0.63253011]
    [0.40546755 0.35522468 0.23930777]
    [0.37774182 0.26117363 0.36108455]
    [0.40079315 0.46998894 0.12921791]]
    >>> sample_series(filtered_population_names, population_labels,
    population_predictions, 4)
    (array(['Diploid_10_ACTGAGCG_TCGACTAG_R1fastqgz',
        'Diploid_10_ATGCGCAG_GCGTAAGA_R1fastqgz',
        'Diploid_10_ACTGAGCG_GCGTAAGA_R1fastqgz',
        'Diploid_10_ACTGAGCG_GAGCCTTA_R1fastqgz'], dtype='<U38'),
    array([0, 1, 2, 2]),
    array([[0.53080315, 0.21029382, 0.25890303],
            [0.40079315, 0.46998894, 0.12921791],
            [0.31415687, 0.17877836, 0.50706477],
            [0.15329386, 0.26029309, 0.58641306]]))
```

3.4 reConstruct - data preparation

3.4.1 get_enriched_series_data

Description:
This function enriches contact matrices of cells in the series by perforeming search in KDTree to find most similar cells in population based on the interphase prediction.

Parameters:

`series_names` (`list[str]`): List of cells names to be enriched.

`series_predictions` (`list[np.ndarray]`): List of prediction np.array vectors.

`population_names` (`list[str]`): List of cells names from whole population.

`population_dir` (`str`): Scool file directory.

`chroms_list` (**Optional** `Optional[list[str]]`, **default=**`None`): List of chromosome names to loaded. Loads all if not specified.

`debug` (**Optional** `Optional[bool]`, **default=**`False`): Prints control info during execution.

Returns:

`list[np.ndarray]` list of enriched n-dim numpy ndarrays with scHiC contact matricies for each cell from the series.

`list[pd.Series]` list of bins description for contact maps for each cell from the series.

Examples:

```
>>> get_enriched_series_data(['Diploid_10_ACTGAGCG_TCGACTAG_R1fastqgz'],
[[0.46271356 0.35939344 0.05374532 0.12414767]],
[['Diploid_10_ACTCGCTA_TCGACTAG_R1fastqgz' ...
],
'../../../data/nagano2017/nagano_1MB_raw.scool',
['chr1', 'chr2', 'chr3', 'chr4', 'chr5'])
    ([array([[0.....]])],
        [    chrom      start        end
        0    chr1           0    1000000
        1    chr1     1000000    2000000
        2    chr1     2000000    3000000
        3    chr1     3000000    4000000
        4    chr1     4000000    5000000
        ..    ...         ...        ...
        844  chr5   148000000  149000000
        845  chr5   149000000  150000000
        846  chr5   150000000  151000000
        847  chr5   151000000  152000000
        848  chr5   152000000  152537259
```

3.4.2 get_series_data

Description:
Extracts data about cells from the series using load_data function.

Parameters:

`series_names` (`list[str]`): List of cells names.

`population_dir` (`str`): Scool file directory containing cells data.

`chroms_list` (**Optional** `Optional[list[str]]`, **default=**`None`): List of chromosome names to loaded. Loads all if not specified.

Returns:

`list[np.ndarray]` list of n-dim numpy ndarrays with scHiC contact matricies for each cell from the series

`list[pd.Series]` list of bins description for contact maps for each cell from the series

Examples:

```
>>> get_series_data(['Diploid_10_ACTGAGCG_TCGACTAG_R1fastqgz'],
'../../../data/nagano2017/nagano_1MB_raw.scool',
['chr1', 'chr2', 'chr3', 'chr4', 'chr5'])
    ([array([[0....]])],
        [    chrom      start        end
        0    chr1           0    1000000
        1    chr1     1000000    2000000
        2    chr1     2000000    3000000
        3    chr1     3000000    4000000
        4    chr1     4000000    5000000
        ..    ...         ...        ...
        844  chr5  148000000  149000000
        845  chr5  149000000  150000000
        846  chr5  150000000  151000000
        847  chr5  151000000  152000000
        848  chr5  152000000  152537259
```

3.4.3 get_supp_contacts

Description:
Extracts required number of random contacts from specified cell's HiC matrix. Since contacts need to be simmetrical most of them will be doubled to the other side of HiC matrix. If you want to determine total number of elements extracted from cell's contact matrix set contacts_num parameter to half of that number.

Parameters:

`supp_cell` (`str`)**:** Cell name.

`contacts_num` (`int`)**:** Number of contacts to be extracted from the cell's HiC matrix.

`population_dir` (`str`)**:** Scool file directory containing cells data.

`chroms_list` **(Optional** `Optional[list[str]]`**, default=**`None`**):** List of chromosome names to loaded. Loads all if not specified.

Returns:

`np.ndarray` HiC matrix with extracted contacts

Examples:

```
>>> get_supp_contacts('Diploid_10_ACTCGCTA_TCGACTAG_R1fastqgz',
100,
'./../../data/nagano2017/nagano_1MB_raw.scool',
['chr1', 'chr2', 'chr3', 'chr4', 'chr5'])
    array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
    >>> np.count_nonzero(
    get_supp_contacts('Diploid_10_ACTCGCTA_TCGACTAG_R1fastqgz',
100,
'./../../data/nagano2017/nagano_1MB_raw.scool',
['chr1', 'chr2', 'chr3', 'chr4', 'chr5']))
```

3.4.4 enrich_hic

Description:
Enriches referenced cell with contacts from support cells. Contacts are added equaly from all support cells and their number is set with a parameter as fraction of referenced cell contact numbers.

Parameters:

`ref_cell` (`str`): Referencial cell name.

`supports` (`list[str]`): List of support cells names.

`population_dir` (`str`): Scool file directory containing cells data.

`chroms_list` (**Optional** `Optional[list[str]]`, **default=**`None`): List of chromosome names to loaded. Loads all if not specified.

`extraction_fraction` (**Optional** `Optional[int]`, **default=**`0.1`): Fraction of oryginal contacts that is supposed to be added

`debug` (**Optional** `Optional[bool]`, **default=**`False`): Prints control info during execution.

Returns:

`list[np.ndarray]` Enriched n-dim numpy ndarray with scHiC contact matrix of referenced cell.

`list[pd.Series]` Bins description for contact map of referenced cell.

Examples:

```
>>> enrich_hic('Diploid_10_ACTGAGCG_TCGACTAG_R1fastqgz',
['Diploid_10_ACTCGCTA_TCGACTAG_R1fastqgz',
'Diploid_10_ACTGAGCG_AAGGCTAT_R1fastqgz'],
'../../../data/nagano2017/nagano_1MB_raw.scool',
                   ['chr1', 'chr2', 'chr3', 'chr4', 'chr5'],
                   debug=True)
    reference contacts num 9782
        support new contacts num 1816
        new total contacts: 11292
    (array([[0...]]),
                chrom      start        end
        0    chr1          0    1000000
        1    chr1    1000000    2000000
        2    chr1    2000000    3000000
        3    chr1    3000000    4000000
        4    chr1    4000000    5000000
        ..    ...        ...        ...
        844  chr5  148000000  149000000
        845  chr5  149000000  150000000
        846  chr5  150000000  151000000
        847  chr5  151000000  152000000
        848  chr5  152000000  152537259
```

3.4.5 matrix_scalling

Description:
Scales down HiC matrix provided by a scale ratio provided by taking a mean of values over a window of size equal to scale.

Parameters:

`matrix` (`np.ndarray`): Oryginal contact matrix to be scaled.

`scale` (`int`): Integer number determining ratio of scaling.

Returns:

`np.ndarray` Scaled down contact matrix.

Examples:

```
>>> print(hic.shape)
    (849, 849)
    >>> hic_scaled = matrix_scalling(hic, 8)
        >>> print(hic_scaled.shape)
    (106, 106)
```

3.4.6 bins_scalling

Description:
Scales down number of bins in the bins description pandas Series provided. Groups bins to achieve desired end number by taking mode of chromosomes assigned of the original bins as well as minimal start and maximal end.

Parameters:

`bins` (`pd.Series`): Original bins description.

`desired_num_bins` (`int`): Number of bins to be returned at the end.

`debug` (**Optional** `Optional[bool]`, **default=**`False`): Prints control info during execution.

Returns:

`pd.Series` Scaled down bins description pandas Series.

Examples:

```
>>> print(len(bin))
    849
        >>> bin_scaled = bins_scalling(bin, 106)
        >>> print(len(bin_scaled))
        106
```

3.4.7 generate_iterations_data

Description:
Scales down number of bins and size of contact matricies of all HiC maps and bins descriptions provided to achieve desired number of verisons. Parameter n controls number of data sets to be reached at the end, where first data set is leaved as original size data. After each scaling remove_diag_plus and normalize_hic functions are applied again.

Parameters:

`hics` (`list[pd.Series]`): Original HiC matricies of cells.

`bins` (`list[pd.Series]`): Original bins descriptions of cells.

`n` (`int`): Number of verisions to be returned at the end.

`p` (**Optional** `Optional[int]`, **default=**`90`): Controls normalize_hic function behaviour.

Returns:

`pd.Series` Scaled down bins description pandas Series.

Examples:

```
>>> generate_iterations_data(hics, bins, 5)
    ([[array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          ...,
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           1.60943791],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 1.60943791, 0.        ,
           0.        ]]),
   array([[0.        , 0.        , 0.2138843 , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.2138843 , 0.        , 0.        , ..., 0.02772589, 0.        ,
           0.        ],
          ...,
          [0.        , 0.        , 0.02772589, ..., 0.        , 0.        ,
           0.15955936],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.15955936, 0.        ,
           0.        ]])],
  [array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          ...,
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.69314718],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.69314718, 0.        ,
           0.        ]]),
   array([[0.        , 0.        , 0.14755518, ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.14755518, 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          ...,
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.78675043],
          [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
           0.        ],
          [0.        , 0.        , 0.        , ..., 0.78675043, 0.        ,
           0.        ]])]],
        [[    chrom      start         end
        0    chr1          0     1000000
        1    chr1    1000000     2000000
        2    chr1    2000000     3000000
        3    chr1    3000000     4000000
        4    chr1    4000000     5000000
        ..    ...        ...         ...
```

```
844  chr5  148000000  149000000
845  chr5  149000000  150000000
846  chr5  150000000  151000000
847  chr5  151000000  152000000
848  chr5  152000000  152537259

[849 rows x 3 columns],
        chrom      start        end
0    chr1          0    6000000
1    chr1    6000000   11000000
2    chr1   11000000   16000000
3    chr1   16000000   21000000
4    chr1   21000000   26000000
..    ...        ...        ...
164  chr5  128000000  133000000
165  chr5  133000000  138000000
166  chr5  138000000  143000000
167  chr5  143000000  148000000
168  chr5  148000000  152537259

[169 rows x 3 columns]],
[    chrom      start        end
0    chr1          0    1000000
1    chr1    1000000    2000000
2    chr1    2000000    3000000
3    chr1    3000000    4000000
4    chr1    4000000    5000000
..    ...        ...        ...
844  chr5  148000000  149000000
845  chr5  149000000  150000000
846  chr5  150000000  151000000
847  chr5  151000000  152000000
848  chr5  152000000  152537259

[849 rows x 3 columns],
        chrom      start        end
0    chr1          0    6000000
1    chr1    6000000   11000000
2    chr1   11000000   16000000
3    chr1   16000000   21000000
4    chr1   21000000   26000000
..    ...        ...        ...
164  chr5  128000000  133000000
165  chr5  133000000  138000000
166  chr5  138000000  143000000
167  chr5  143000000  148000000
168  chr5  148000000  152537259

[169 rows x 3 columns]]])
```

3.4.8 check_iterations_setup

Description:
For the cell's index in the hic_scales list, prints number of bins and number of non zero values in the hic matrix and plots each version of hic matrix of this cell. Allowed values of orientation parameter are 'Horizontal' and 'Vertical'.

Parameters:

`hics_scales (list[str])`: List of verisions of hic matrix.

`cell_idx` (`int`)**:** Index of cell that is to be checked from hic_scales list.

`orientation` (**Optional** `Optional[str]`, **default=**`'Horizontal'`)**:** Orientation of output plot.

Returns:

`list[np.ndarray]` list of enriched n-dim numpy ndarrays with scHiC contact matricies for each cell from the series.

`list[pd.Series]` list of bins description for contact maps for each cell from the series.

Examples:

```
>>> check_iterations_setup(hics_scales, 1)
    computational load:
        iteration 1
        number of particles: 169
        number of bonds: 5154
        iteration 2
        number of particles: 849
        number of bonds: 22498
```

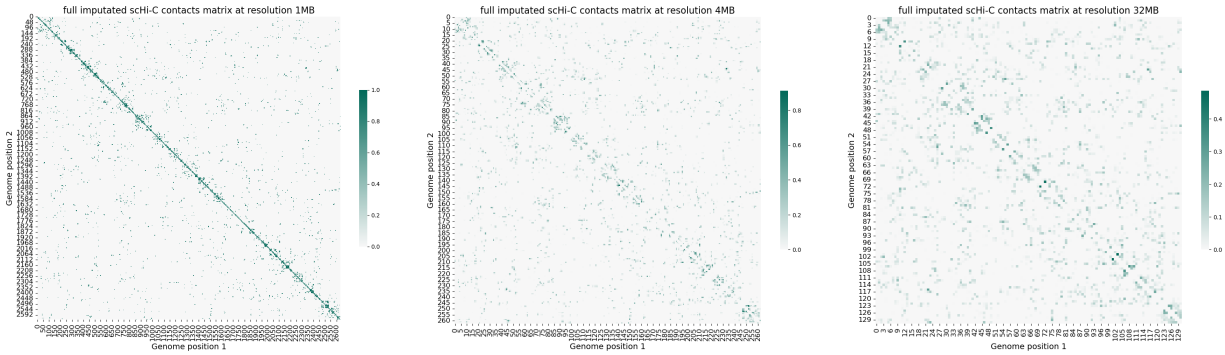


Figure 3: Matrix scalling example.

3.5 reConstruct - simulation

3.5.1 generate_new_particles

Description:
Calculates new set of particles positions to achieve description of bins provided by next_bin while staying within chains position taken from last frame of simulation record provided by trajectory_dir. New particles are spread evenly along chains length.

Parameters:

`trajectory_dir` (`str`)**:** String specifying location of .gsd file holding in it end state of a simulation which resolution user wants to expand.

`next_bin` (`pd.Series`)**:** Bins description which will determine number of particles returned

`debug` (**Optional** `Optional[bool]`, **default=**`False`)**:** Prints control info during execution.

Returns:

`list[np.array]` List of 3 dimensional np.arrays holding new positions of particles.

Examples:

```
>>> generate_new_particles('./test_frame.gsd', bins_scales[0][0], debug=True)
    chromosom_legend {0: 'chr1', 1: 'chr2', 2: 'chr3', 3: 'chr4', 4: 'chr5'}
        chrom generated: chr1
        chrom_start_num: 39
        chrom_end_num: 198
        chrom_add_num: 159
        old_idx [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13
    14 15 16 17 18 19 20 21 22 23
        24 25 26 27 28 29 30 31 32 33 34 35 36 37]
        old_idx num 38
        new_idxs [ 0  0  0  0  0  1  1  1  1  2  2  ...]
        new_idxs num 197
        starting from [[ 0.0000000e+00  0.0000000e+00  0.0000000e+00]
        [ 0.0000000e+00  0.0000000e+00  0.0000000e+00]
        [ 0.0000000e+00  0.0000000e+00  0.0000000e+00]
        [ 0.0000000e+00  0.0000000e+00  0.0000000e+00]
        [ 0.0000000e+00  0.0000000e+00  0.0000000e+00]
        [-1.7052641e+00  5.7458645e-01 -3.0677056e+00]
        [-1.7052641e+00  5.7458645e-01 -3.0677056e+00]
        ...
        [60.49659337 14.14804498 45.72595499]
        [61.00875445 14.45202941 46.11389353]
        [61.47748046 14.72086156 46.30976203]
        [61.75637381 14.83606192 45.66621001]]
        Output is truncated. View as a scrollable
        element or open in a text editor. Adjust cell output settings...
        array([[ 0.        ,  0.        ,  0.        ],
                [-0.31627893,  0.10656977, -0.56897386],
                [-0.63255787,  0.21313953, -1.13794771],
                ...,
                [61.00875445, 14.45202941, 46.11389353],
                [61.47748046, 14.72086156, 46.30976203],
                [61.75637381, 14.83606192, 45.66621001]])
```

3.5.2 generate_initial_positions

Description:
Generates initial positions of particles to be used in simulation initiation by using random walk.

Parameters:

`particles_count (int)`: Number of particles to be generated

`radius (int)`: Size of the step when performing random walk. Will determine distance between generated positions.

`box_size (int)`: Constraint on positions generated ensuring no particle will be given position outside of the simulation box.

Returns:

`list[np.array]` List of 3 dimensional np.arrays holding initial positions of particles.

Examples:

```
>>> generate_initial_positions(169, 1, 50)
    array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-5.72993301e-01, -9.16848623e-01, -5.40761736e-01],
       [ 7.62070111e-02, -2.05794635e-01, -6.13924149e-01],
       [-6.81294849e-01,  5.99844209e-01,  3.60949628e-02],
       [ 1.63254061e-01,  1.54804675e+00, -6.80280606e-02],
```

```
       [-7.19805624e-01,  1.82536159e+00, -8.99686964e-01],
       [-1.35698088e+00,  1.44191747e+00, -1.06326082e+00],
       [-5.27316493e-01,  1.71036633e+00, -1.75043193e+00],
       [ 2.65728128e-01,  2.13923773e+00, -2.45423359e+00],
       [-1.31685503e-01,  2.67329759e+00, -2.90523244e+00],
       [ 8.60127891e-02,  2.75981073e+00, -2.94916762e+00],
       [-1.03529561e-01,  3.14450165e+00, -3.64494361e+00],
       [ 2.88306448e-01,  3.49293476e+00, -4.02396203e+00],
       [-6.93212701e-01,  3.63064657e+00, -4.61636354e+00],
       [ 1.93876078e-01,  2.95197476e+00, -4.11671417e+00],
       [ 6.59374221e-01,  3.17923378e+00, -4.39934160e+00],
       [ 1.12749628e+00,  3.92942818e+00, -3.52535340e+00],
       [ 2.34325781e-01,  4.59357635e+00, -2.75406365e+00],
       [ 9.18592357e-01,  5.40573265e+00, -3.04404011e+00],
       [ 1.10497319e+00,  4.63750138e+00, -3.99706130e+00],
       [ 1.70790741e+00,  4.44501844e+00, -4.66123650e+00],
       [ 2.55263001e+00,  4.21102206e+00, -5.51010015e+00],
       [ 2.27172913e+00,  4.08972764e+00, -6.00980706e+00],
       [ 3.23212767e+00,  4.85937866e+00, -5.43301697e+00],
       [ 3.96052339e+00,  5.84950415e+00, -4.45938346e+00],
        ...
       [ 8.93377412e+00, -1.41604450e+01, -3.29538790e+00],
       [ 9.61538778e+00, -1.50483063e+01, -2.52155734e+00],
       [ 9.12140479e+00, -1.56316652e+01, -1.81626101e+00],
       [ 8.66009048e+00, -1.65017794e+01, -1.00518598e+00],
       [ 9.31116976e+00, -1.65338110e+01, -1.38459180e-01]])
```

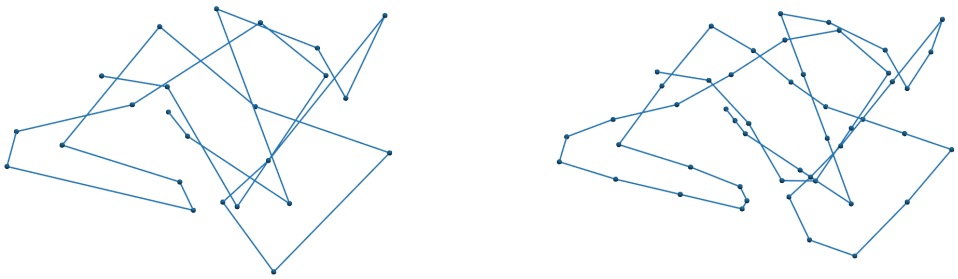


Figure 4: Exemplary picture of end state of reconstruction iteration (left), with picture of the same structure after new particles positions generation (right).

3.5.3 frame_initiation

Description:
Initializes first frame of the simulation. Defines particles types and positions as well as forces in the simulation(chain_force, contact_force, colision_force, colision_force_weak). When debug parameter is set to False saves frame to the .gsd file and return list of hoomd.md objects specifying forces taking place in the simulation.

Parameters:

`matrix (np.ndarray):` HiC matrix on which simulation will be based.

`bins (pd.Series):` Bins description on which simulation will be based.

`frame_name (str):` Filename of .gsd in which frame will be saved to.

`new_particles_position` **(Optional** `Optional[list[np.array]]`**, default=**`None`**):** Positions in which particles will be placed. When no positions specified generate_initial_positions function will be used.

`debug` **(Optional** `Optional[bool]`**, default=**`False`**):** Prints control info during execution.

Returns:

`list[np.ndarray]` list of enriched n-dim numpy ndarrays with scHiC contact matricies for each cell from the series.

`list[pd.Series]` list of bins description for contact maps for each cell from the series.

Examples:

```
>>> frame_initiation(hics_scales[0][1], bins_scales[0][1],
'./test_frame.gsd', debug=True)
    contact_count: 4854
        len(bins.chrom): 5
        particles:
        num  169
        postitions num  169
        types  ['chr1', 'chr2', 'chr3', 'chr4', 'chr5']
        types num  169
        bonds:
        num  5018
        types  ['chr1-chr1', 'chr2-chr2', 'chr3-chr3',
    'chr4-chr4', 'chr5-chr5', 'contact']
        typeid [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        chromosom_legend {'chr1': 0, 'chr2': 1, 'chr3': 2,
    'chr4': 3, 'chr5': 4}
        unique types num [0, 1, 2, 3, 4, 5]
        types num  5018
        groups num  5018

        [<hoomd.md.bond.Harmonic at 0x7f2cddbc3f40>,
        <hoomd.md.bond.Harmonic at 0x7f2cddbd4f70>,
        <hoomd.md.pair.pair.Gaussian at 0x7f2cddbc2f10>,
        <hoomd.md.pair.pair.Gaussian at 0x7f2cddbc2f10>]
```

3.5.4 run_sim

Description:
Runs simulation. After loading initiatory frame from .gsd file specified in state_dir will create hoomd.Simulation from it and add forces within hoomd.md.Integrator. Uses langevin model with kT=1.0 and time step of 0.005.

Parameters:

`state_dir` **(**`str`**):** Path to the file holding initiated before hand first frame of the simulation.

`forces` **(**`list[hoomd.md.bond.Harmonic | hoomd.md.pair.Gaussian]`**):** List of forces to be added to the simulation.

`simple` **(Optional** `Optional[bool]`**, default=**`None`**):** When set to True will perform simulation without forces manipulations and for reduced number of 8e4 steps.

Returns:

`None`

Examples:

```
>>> run_sim('./test_frame.gsd', forces)
```

3.5.5 perform_single_reconstruction

Description:
Performs single round of reconstruction.

Parameters:

`dir_str` (`str`): Directory in which simulations records will be saved.

`hics_scales` (`list[np.ndarray]`): List of HiC matrices used in each iteration.

`bins_scales` (`list[pd.Series]`): List of bins descriptions used in each iteration.

`stop_early` (**Optional** `Optional[int]`, **default=**`0`): Limits number of iterations stopping reconstruction earlier.

`save_each_iteration` (**Optional** `Optional[bool]`, **default=**`False`): Specifies weather to save all iterations .gsd file or only final one.

`save_screenshots` (**Optional** `Optional[bool]`, **default=**`False`): Specifies weather to save images of each iteration resoult.

`visualize_result` (**Optional** `Optional[bool]`, **default=**`False`): Specifies weather to visualize last iteration result at the end of reconstruction.

Returns:

`None`

Examples:

```
>>> perform_single_reconstruction('test5/', series_hics_scales[3],
                                  series_bins_scales[3], stop_early = 1)
    running iteration: 4
    bins num 42
    hic shape (42, 42)
    running sim...

    running iteration: 3
    bins num 56
    hic shape (56, 56)
    running sim...

    running iteration: 2
    bins num 84
    hic shape (84, 84)
    running sim...

    running iteration: 1
    bins num 169
    hic shape (169, 169)
    running sim...
    visualising results...
```

3.5.6 perform_many_reconstructions

Description:
Performs many runs of the same reconstruction.

Parameters:

`dir_str` (`str`): Directory in which simulations records will be saved.

`hics_scales` (`list[np.ndarray]`): List of HiC matrices used in each iteration.

`bins_scales` (`list[pd.Series]`): List of bins descriptions used in each iteration.

`runs_num` (`int`): Controls number of runs.

`stop_early` (**Optional** `Optional[int]`, **default=**`0`): Limits number of iterations stopping reconstruction earlier.

`logs` (**Optional** `Optional[str]`, **default=**`'runs'`): Set to 'runs' or 'iterations' to control printing of steps taking place. Default value is 'runs'.

Returns:

`None`

Examples:

```
>>> perform_many_reconstructions('test6/', series_hics_scales[2],
                                 series_bins_scales[2], runs_num = 5,
                                 stop_early = 1)
    running run 0
    running iteration: 4
    bins num 42
    hic shape (42, 42)
    running iteration: 3
    bins num 56
    hic shape (56, 56)
    running iteration: 2
    bins num 84
    hic shape (84, 84)
    running iteration: 1
    bins num 169
    hic shape (169, 169)

    running run 1
    running iteration: 4
    bins num 42
    hic shape (42, 42)
    running iteration: 3
    bins num 56
    hic shape (56, 56)
    running iteration: 2
    bins num 84
    hic shape (84, 84)
    ...
    hic shape (84, 84)
    running iteration: 1
    bins num 169
    hic shape (169, 169)
```

3.6 reConstruct - review

3.6.1 remove_contact_bonds

Description:
Will remove contact bonds from all frames of the simulation record .gsd file. Usefull for visualizations using 3rd party software.

Parameters:

`gsd_trajectory` (`str`): Path to the .gsd file containing simulation record.

Returns:

`None`

Examples:

```
>>>     remove_contact_bonds('./test7/frame_4_traj.gsd')
        File saved to the ./test7/frame_4_traj_no_contact.gsd
```

3.6.2 visualize_sim

Description:
Visualizes simulation end state from last frame of the trajectory_dir .gsd file specified. Uses pyvista package and allows for selection of chromosomes, visualization settings and whether to save its picture.

Parameters:

`trajectory_dir` (`str`):

`no_contacts` (**Optional** `Optional[bool]`, **default**=`True`):

`screenshot` (**Optional** `Optional[bool]`, **default**=`False`):

`no_visualize` (**Optional** `Optional[bool]`, **default**=`False`):

`chroms` (**Optional** `Optional[list[str]]`, **default**=`None`):

`chain_width` (**Optional** `Optional[int]`, **default**=`5`):

`contact_width` (**Optional** `Optional[int]`, **default**=`0.25`):

`particle_size` (**Optional** `Optional[int]`, **default**=`10`):

`debug` (**Optional** `Optional[bool]`, **default**=`False`):

Returns:

`None`

Examples:

```
>>> visualize_sim('./test_frame_traj.gsd', screenshot=True)
```

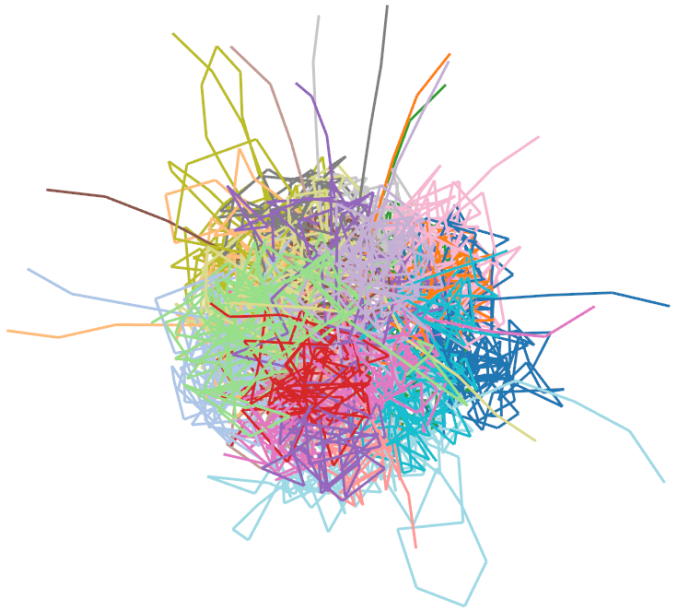


Figure 5: Example of Nagano based reconstruction with final resolution at 1Mb.

3.6.3 inspect_gsd

Description:
Prints out information about particles from last simulation frame within specified .gsd file.

Parameters:

`gsd_traj` **(`str`):** Directory of a .gsd file to be inspected.

Returns:

`None`

Examples:

```
>>> inspect_gsd('./test_frame_traj.gsd')
    chromosom_legend: {0: 'chr1', 1: 'chr2', 2: 'chr3', 3: 'chr4', 4: 'chr5'}
        particles num 169
        particles_position: [[ -9.152195   14.470503    4.625657 ]
        [ -8.966829   13.731618    4.279521 ]
        [ -9.085836   14.734003    3.9680572]
        [-10.579327   13.710347    4.338241 ]
        [-10.103192   13.565733    4.786075 ]]
        particles_types: ['chr1', 'chr2', 'chr3', 'chr4', 'chr5']
        particles_types_id: [0 0 0 0 0]
        particles_chrom: ['chr1', 'chr1', 'chr1', 'chr1', 'chr1']

        bonds num 5018
        bonds_particles: [[0 1]
        [1 2]
        [2 3]
        [3 4]
        [4 5]]
        bonds_types: ['chr1', 'chr2', 'chr3', 'chr4', 'chr5', 'contact']
        bonds_types_id: [0 0 0 0 0]
```

```
        bonds_legend: {0: 'chr1', 1: 'chr2', 2:
        'chr3', 3: 'chr4', 4: 'chr5', 5: 'contact'}
        bonds_legen_test {0: 'chr1', 1: 'chr2', 2: 'chr3',
        3: 'chr4', 4: 'chr5', 5: 'contact'}
        bonds_chrom: ['chr1', 'chr1', 'chr1', 'chr1', 'chr1']
        no_contact_check [False False False ...  True  True  True]
        bonds_particles_no_contact
        [array([0, 1], dtype=uint32), array([1, 2], dtype=uint32),
        array([2, 3], dtype=uint32), array([3, 4],
        dtype=uint32), array([4, 5], dtype=uint32)]
        bonds_chrom_no_contact ['chr1', 'chr1', 'chr1', 'chr1', 'chr1']
        bonds_particles_contact
        [array([0, 2], dtype=uint32), array([0, 3], dtype=uint32),
        array([0, 4], dtype=uint32),
        array([0, 5], dtype=uint32),
        array([0, 6], dtype=uint32)]
        bonds_chrom_contact
        ['contact', 'contact', 'contact', 'contact', 'contact']
```

3.6.4 get_aligned_structure

Description:
Aligns structure within .gsd file to the reference postion using Kabsch algorithm.

Parameters:

`gsd_traj_to_align` (`str`): Directory of a gsd file.

`reference_positions` (`np.ndarray`): 3xn ndarray holding postitions of reference structure particles.

Returns:

`np.ndarray` Optimaly rotated and translated postitions of particles after alignment.

Examples:

```
>>> get_aligned_structure('./nagano_rmsd/run_1_frame_1_traj.gsd',
                          ref_strucutre)
array([[ 1.62143703e-03,  1.43938386e+00,  1.70843855e-01],
       [ 4.42002922e-01,  9.22812879e-01, -7.73531377e-01],
       [-7.46729910e-01,  1.18272567e+00, -1.15268409e+00],
       [-1.75923761e-02,  3.04058075e-01, -5.96112311e-01],
       [ 4.22458202e-01,  1.35571098e+00, -1.45108068e+00],
       [ 7.34163284e-01,  1.85640788e+00, -4.97791767e-02],
       [-6.34560108e-01,  1.88624573e+00, -4.14429186e-03],
       [ 4.80610952e-02,  2.46881413e+00, -6.17990732e-01],
       [-5.99977851e-01,  1.43865776e+00, -7.13989377e-01],
       [ 5.21241784e-01,  8.80188882e-01, -1.22290540e+00],
       [ 6.62433028e-01,  1.47768342e+00, -1.97373271e-01],
       [ 6.09422207e-01,  2.62223892e-02, -2.30374575e-01],
       [ 8.57710242e-01,  6.49624467e-01, -1.04118204e+00],
       [ 7.41387427e-01,  1.45418561e+00,  7.02167228e-02],
       [-4.02550250e-02,  9.89303946e-01,  4.14790452e-01],
       [ 3.63107592e-01, -2.04363301e-01, -1.22636996e-01],
       [-6.55005872e-01,  3.18996161e-01, -1.99320152e-01],
       [-6.46436632e-01,  1.20663559e+00, -9.75228429e-01],
       [-4.73645389e-01,  5.92496172e-02, -1.50494063e+00],
       [-1.22011089e+00,  7.22957134e-01, -1.46668613e+00],
       [-2.75297046e-01,  1.62346148e+00, -2.19041109e+00],
       [-7.44748652e-01,  1.73683572e+00, -1.07604635e+00],
       [-3.35611433e-01,  4.90552753e-01, -1.92115259e+00],
       [ 5.93715906e-02,  1.60838807e+00, -1.96075797e+00],
```

```
       [ 8.21302176e-01,  1.94787908e+00, -1.41415322e+00],
...
       [-1.42637506e-01, -7.07066357e-01,  2.06233263e+00],
       [ 8.75487089e-01, -1.36590755e+00,  1.92228436e+00],
       [-3.78580868e-01, -1.10006785e+00,  1.49170744e+00],
       [-1.01297237e-01,  2.83424258e-01,  9.99071538e-01],
       [-4.49965477e-01, -4.52558011e-01, -6.73871429e-04]], dtype=float32)
```

3.6.5 get_centered_structure

Description:
Calculates postion of particles from .gsd structure after being centered to the (0,0,0) point.

Parameters:

`gsd_traj` (`str`)**:** Directory of a gsd file.

Returns:

`np.ndarray` Postitions of particles after centering.

Examples:

```
>>> get_centered_structure('./nagano_rmsd/run_0_frame_1_traj.gsd')
array([[-0.4473364 ,  2.0602455 , -1.121573  ],
       [ 0.45622253,  1.5839031 , -0.5479775 ],
       [-0.17814198,  0.24082708, -0.53601545],
       [-0.52943325, -0.53362083, -1.2180109 ],
       [ 0.34416944,  0.05312443, -1.9454908 ],
       [-0.34118748,  1.1826465 , -1.6515245 ],
       [ 1.1468173 ,  1.271433  , -1.2586889 ],
       [ 1.0712112 ,  0.28182554, -1.1126066 ],
       [ 0.9567336 ,  1.3858814 , -1.5742903 ],
       [ 0.24181977,  2.0227196 , -0.7861956 ],
       [-0.25937733,  0.90937114, -1.1740656 ],
       [ 0.65947527,  0.282614  , -0.68194187],
       [ 0.88874424,  0.9778516 ,  0.11983812],
       [ 0.3647266 ,  1.9112248 , -0.5452133 ],
       [-0.3940391 ,  1.1334629 , -0.9187726 ],
       [-0.7059277 ,  2.1761625 ,  0.22949219],
       [-0.96230495,  1.0952504 , -0.30846035],
       [-0.23567039,  1.3875872 ,  0.4776423 ],
       [-0.7739358 ,  0.9526913 , -0.5589328 ],
       [-0.41030258,  0.8660705 , -1.5868237 ],
       [ 0.5084163 ,  0.6166518 , -0.3237368 ],
       [-0.31579572,  1.5593971 ,  0.1484232 ],
       [ 1.209608  ,  1.5266508 ,  0.7997948 ],
       [ 0.7820178 ,  0.7929921 , -0.18128097],
       [ 1.3587208 ,  1.6190186 , -0.32116938],
...
       [ 0.59842503, -1.932611  , -0.81298196],
       [ 1.3838544 , -1.5882578 ,  0.27190435],
       [ 0.9899758 , -0.67888427,  1.2646391 ],
       [ 0.05472907, -0.15567446,  0.27302122],
       [-0.5732895 , -0.7074845 ,  1.1585584 ]], dtype=float32)
```

3.6.6 calculate_rmsd

Description:
Calculates root mean square deviation between two structures particles.

Parameters:

`positions_1` (`np.ndarray`): First strcture particles positions.

`positions_2` (`np.ndarray`): Second strcture particles positions.

Returns:

`float` Root mean square deviation between two structures particles.

Examples:

```
>>> calculate_rmsd(ref_strucutre, alig_strucutre)
1.3634467130030685
```

3.6.7 check_structures_rmsd

Description:
Calculates root mean square deviation for set of structures between each of those structures and mean strucure based on them. Uses Kabsch algorithm to align all structures.

Parameters:

`gsd_trajs` (`list[str]`): List of .gsd files.

Returns:

`list[float]` List of root mean square deviation for each structure.

Examples:

```
>>> check_structures_rmsd(structures)
[1.0883503887457244,
 1.226046156228531,
 1.2399511427191139,
 1.414559489329183,
 1.0188597957689096,
 1.1689273131559086,
 1.1860160572684715,
 1.0173071881928553,
 1.3020940889595223,
 1.0678061777160002,
 1.1618736183869356,
 1.2353679865107023,
 1.1845022812543988,
 1.4870578031397714,
 1.139476483909423,
 1.0830905499301324,
 1.0896910135465745,
 1.1153874890520632,
 1.2751206539756772,
 1.153490764955951,
 1.2593302109934554,
 1.1162473544512173,
 1.1282030163369499,
 1.0829797977300928,
 1.2173346367798918,
...
 1.4488858029854934,
 0.9796518257840011,
 1.1526770904892438,
 1.0898953201277846,
 1.067808037931412]
```