HiFive Documentation

Release 2.0.0

Michael Sauria

October 20, 2014

Contents

Contents:

CHAPTER 1

Introduction

HiFive is a Python package for normalization and analysis of chromatin structural data produced using either the 5C of HiC assay. This library contains tools for handling all steps after mapping of reads.

1.1 Organization of the HiFive package

The HiFive package is split into several modules, each serving a specific purpose.

Functionality	Module
Restriction enzyme information	Fragment / Fend
Read counts and orientations	FiveCData / HiCData
Model parameters and filtering	FiveC / HiC
Boundary index	BI
Plotting functions	plotting

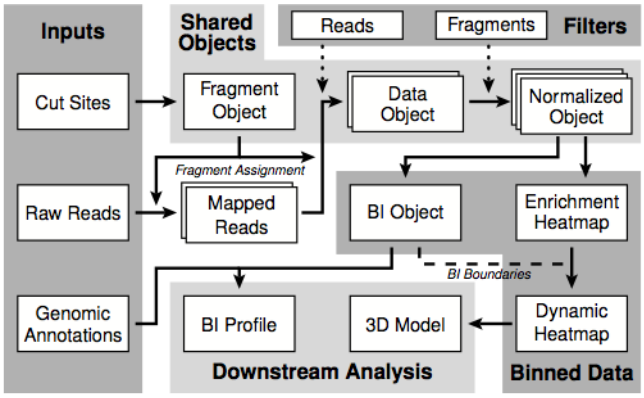
The classes Fragment, Fend, FiveCData, HiCData, FiveC, HiC, and BI are all available from the top level of the HiFive namespace and can be imported using:

```
from hifive import *
```

at the beginning of the Python program. However, in order to prevent namespace pollution, you may also simply use import hifive.

HiFive is organized into a hierarchy of data structures. Each structure represents a set of data that may be shared with any number of structures higher in the hierarchy, thus eliminating redundency of information. For example, a Fragment object which contains information about the fragments being interogated in a 5C experiment can be used for all replicates and conditions that use the same primer scheme. Likewise, a HiCData object which contains all of the mapped read information for a specific HiC experiment can be used for multiple analyses with different parameter values. This helps reduce the space these data occupy as well as reduce the time to run mutiple analyses since each object need only be created once.

The organization of structures is as follows:


Inputs
Shared Objects
Reads
Fragments
Filters
Cut Sites
Fragment Object
Data Object
Normalized Object
Fragment Assignment
Raw Reads
Mapped Reads
BI Object
Enrichment Heatmap
BI Boundaries
Genomic Annotations
BI Profile
3D Model
Dynamic Heatmap
Downstream Analysis
Binned Data

CHAPTER 2

Getting started

2.1 Installing HiFive

HiFive can bo obtained from HiFive using the following command:

```
> hg clone https://bitbucket.org/bxlab/hifive/
```

or alternatively, download a snapshot of the repository using the following commands:

```
> wget https://bitbucket.org/bxlab/hifive/downloads/hifive_v2.0.tar.bz2
> tar -xjf hifive_v2.0.tar.bz2
```

HiFive depends on a few packages and has several others that extend its functionality.

2.1.1 Required Packages

- Scipy
- Numpy
- h5py

2.1.2 Recommended Packages

- Pysam
- Pyx
- PIL
- mpi4py
- mlpy

To install HiFive, simply enter the directory that the repository was cloned or downloaded to and use the following command:

```
> python setup.py install
```

If you wish to install HiFive in a location other than the default, you can use the prefix option:

```
> python setup.py install --prefix=/your/desired/path
```

2.2 Installing Documentation

In order to build HiFive's documentation locally, you need to execute the following command:

```
> cd doc
> make html
```

This will create the documentation suitale for viewing with a web browser. In order to create a pdf version of the documentation, simply run the following:

```
> cd doc
> make latexpdf
```

CHAPTER 3

A Brief 5C Tutorial

In order to perform analysis with HiFive, you must construct a series of objects, each one relying on the previous one. This tutorial will walk you through the process for doing a basic analysis of 5C data.

3.1 Creating a Fragment object

In order to use HiFive to process 5C data, the first thing that you need to do is create a Fragment object. The Fragment object is an set of information contained in an HDF5 file. The object contains information about fragment sizes as determined by a series of boundary coordinates and orientations given in a BED formatted file, as well as an index for converting between string-based region names and numerical identifiers. The user may also include information about the genome and restriction enzyme, although these are optional and do not affect the functionality.

To create the Fragment object, you must provide a bed file in which each line contains the up- and downstream boundaries of a fragment and the strand to which this fragment belongs. Fragment names are also stored from the BED file but are not required. Finally, groups of fragments can be separated into distinct regions which will be handled separately in downstream normalization and analysis. Fragments occuring on different chromosomes are always assigned to different regions. In addition, the user has the option of either explicitly specifying a list of regions or giving a minimum separation distance between fragments that is used to partition fragments into distinct regions.

To create a basic Fragment object, use the following command:

```
import hifive
fragment = hifive.Fragment(out_filename, mode='w')
fragment.load_fragments(bed_filename, genome_name='MM9', re_name='HindIII')
fragment.save()
```

In this case, the 'out_filename' specifies the location to save the Fragment object to and should end with the '.hdf5' extension. The 'bed_filename' contains the fragment boundaries, primer names, and strand information in BED format. The 'genome_name' and 're_name' are option strings that may be passed. In this case, the regions would be automatically determined using the default minimum distance between regions of 1 Mb. This could set by specifying a value with the keyword 'minregionspacing'.

In order to specifiy how to partion fragments into regions, we could use the following commands:

```
import hifive
fragment = hifive.Fragment(out_filename, mode='w')
fragment.load_fragments(bed_filename, regions=[['chr1', start1, stop1], ['chr2', start2, stop2]])
fragment.save()
```

This would create two regions that would include all of the fragments between coordinate values start1 - stop1 and start2 - stop2. When regions are explicitly given, the 'minregionspacing' variable is ignored.

3.2 Creating a FiveCData object

In order to create a 5C dataset, you first need to have created an appropriate Fragment object. You can create the FiveCData object either from a file containing primer pairs and their observed counts or directly from mapped data in BAM format. If using counts data to create the dataset, the format should be:

```
primer1   primer2 #_observed
```

where values are separated by tabs and '#_observed' is an integer. With either format the primer names must exacly match those in the BED file used to create the Fragment object.

To create the 5C dataset, you can run the following commands:

```
import hifive
data = hifive.FiveCData(out_filename, mode='w')
data.load_data_from_counts(fragment_filename, [counts1.txt, counts2.txt])
data.save()
```

In this case, 'out_filename' specifies the location to save the FiveCData object to and should end with the '.hdf5' extension. The 'fragment_filename' value is the location of the appropriate Fragment object. Multiple files containing counts data may be passed to the function as a list or, if only a single counts file is needed, it may be passed as a string. In order to load data from a set of BAM files, a similar procedure is used:

```
import hifive
data = hifive.FiveCDataset(out_filename, mode='w')
data.load_data_from_bam(fragment_filename, [bam_prefix1, bam_prefix2])
data.close()
```

In this case, the only difference is that prefices are passed instead of complete file names. The prefices should be the entire file path up until the strand specifier such that the two file names are created by appending either '_1.bam' or '_2.bam' to the prefix. Like the function for counts data, if only a single prefix is needed it may be passed as a string.

> Note: The FiveCData object can now be used by multiple analyses of this sample and does not need to be created separately for each one.

3.3 Creating a FiveC analysis object

The 5C analysis object, FiveC, contains links to a FiveCData and Fragment object, information about which fragments to include in the analysis, model parameters, and learned model values. This is the standard way of working with 5C data in HiFive and this object will be used for learning the model, extracting portions of data, plotting, and downstream analysis.

To create a FiveC object, you can use the following commands:

```
import hifive
fivec = hifive.FiveC(out_filename, 'w')
fivec.load_data(data_filename)
fivec.save()
```

In this case, 'out_filename' specifies the location to save the FiveC object to and should end with the '.hdf5' extension. The 'data_filename' value is the location of the appropriate FiveCData object.

Warning: Becauase data and fragment data are stored in their own objects, each object keeps track of the location of its dependents through relative file names. This means that links between them will break if the relative pathway is changed.

3.4 Filter 5C fragments

Prior to modeling the data, you need to filter out fragments that have few valid reads mapped to them. HiFive uses an iterative filtering approach such that only when all fragments satisfy a user-defined minimum number of valid interactions does the filtering process cease.

To filter fragments, you can use the following commands:

```
import hifive
fivec = hifive.FiveC(fivec_filename)
fivec.filter_fragments(mininteractions=25)
fivec.save()
```

In this case, 'fivec_filename' is a previously saved FiveC analysis object. No value was passed to mode, since it defaults to 'r' for read. This loads the data from a previously created FiveCData object. In order for changes to be kept to a FiveC object, it must be written to file using the save command.

3.5 Find 5C distance function

HiFive approximates the distance-signal relationship using a power-law regression such that the log of the distance between the midpoints of two fragments and the log of their observed interactions. To do an initial estimate of this function, you can use the following command:

```
fivec.find_distance_parameters()
```

3.6 Learn 5C normalization parameters

In order to learn the correction model for 5C data, HiFive uses two rounds of gradient descent, one with constant learning rate (the 'burn-in' phase) and the second with a linearly decreasing learning rate (the 'annealing' phase). In addition, HiFive can recalculate the distance function parameters periodically using the correction-adjusted interaction values. Finally, HiFive limits which interactions it uses to learn the model parameters to those that fall within a user-specified maximum interaction distance.

To learn 5C corrections using the modeling approach, you can use the following command:

```
fivec.find_fragment_corrections(display=100,
                                maxdistance=0,
                                learningrate=0.01,
                                burnin_iterations=5000,
                                annealing_iterations=10000,
                                recalculate_distance=100)
```

In the above call, 'maxdistance' is set to zero, indicating that there is no upper limit on interaction distance to be used for learning model parameters. The 'recalculate_distance' parameters specifies how many iterations to wait before recalculating the distance parameters. The 'learningrate' specifies what percentage of the gradient to apply towards value updates. One last value passed to the function in 'display', which specifies how many iterations should pass before updating the display (via STDERR). This can also be set to zero to not display the progress.

3.7 Approximate 5C normalization parameters

HiFive also offers an approximation approach for learning correction values. The primary differences to the correction model from the user's perspective are a single learning phase and a lack of learning rate. The approximation

learning approach can still recalculate the distance function parameters periodically.

To learn 5C corrections using the approximation approach, you can use the following command:

```
fivec.find_fragment_corrections(iterations=1000,
                                recalculate_distance=100,
                                remove_distance=True)
```

In the above call, the 'remove_distance' argument specifies whether to remove the distance-dependent portion of the signal prior to approximating correction values. For best results, this should set to true (its default value).

3.8 Generating a heatmap

In order to immediately make use of data, `HiFive` allows you to pull data from a regions and create a heatmap. The data can be returned unbinned, binned using a fixed-width bin size, or binned using boundaries passed by the user. There are several options for the format the data can be passed back in. Please refer to the `cis_heatmap` function for more details. There are also several options for transformations to the data. These are used to remove the distance-dependence signal, fragment bias, both, or to return only the predicted signal. In this example, we'll get a set of data from an entire region binned into 10 Kb bins as follows:

```
heatmap = fivec.cis_heatmap(region=1,
                            binsize=10000,
                            arraytype='compact',
                            datatype='enrichment')
```

In the above call, 'enrichment' specifies to find the observed counts and expected counts, which includes the distance-dependence and fragment bias values. The observed counts are in the first index of the last dimension of the returned array, the expected counts are in the second index of the last dimension. 'compact' specifies a rectangular array where the first axis is the forward primers and the second axis is the reverse primers. 'region' refers to the region index given by `HiFive`. To find out more details about that region, we could do the following:

```
fivec.frags['regions'][1]
```

This returns the region's chromosome, starting fragment, stopping fragment (first fragment outside the region), starting coordinate and stopping coordinate.

3.9 Plotting a heatmap

In order to visualize the heatmap we just produced, `HiFive` has several plotting functions that take different shaped arrays. The function called needs to match the array produced. However, in this case, the 5C compact array is compatible with the `plot_full_array` function, so we'll use that as follows:

```
img = hifive.plotting.plot_full_array(heatmap, symmetric_scaling=True)
img.save(out_fname)
```

In calling the function, we pass the heatmap and that would be sufficient. There are, however, additional options. For example, 'symmetric_scaling' specifies whether the color scale should be expanded to run from the minimum value to the maximum (False) or so that the maximum absolute value determine both upper and lower color bounds. The image returned is a PIL image of type 'png'.

Note: The next thing on the todo list is write wrappers within the `FiveC` and `HiC` classes for running plotting through the analysis objects themselves.

CHAPTER 4

A Brief HiC Tutorial

In order to perform analysis with HiFive, you must construct a series of objects, each one relying on the previous one. This tutorial will walk you through the process for doing a basic analysis of HiC data.

4.1 Creating a Fend object

In order to use HiFive to process HiC data, the first thing that you need to do is create a fragment-end (Fend) object. The Fend object is an set of information contained in an HDF5 file. The object contains information about the fragments created by digestion of a genome by a specific restriction enzyme (RE) as well as an index for converting between string-based chromosome names and numerical identifiers. The user may also include information about the genome and restriction enzyme, although these are optional and do not affect the functionality.

To create a Fend object, you must provide the location of RE fragments. This information is supplied in the form of either a 'fend' file with a format compatible with HiCPipe or a BED-formatted file containing fragment boundaries produced by RE digest or locations of the RE cut sites.

To create a basic Fend object, use the following command:

```
import hifive
fend = hifive.Fend(out_filename, mode='w')
fend.load_fends(RE_data_filename, genome_name='MM9', re_name='HindIII')
fend.save()
```

In this case, the 'out_filename' specifies the location to save the Fend object to and should end with the '.hdf5' extension. The 'RE_data_filename' contains the RE fragment boundaries in one of the formats described above. The 'genome_name' and 're_name' are option strings that may be passed.

Note: The Fend object can now be used by any experiment that relies on the same genome / restriction enzyme combination and does not need to be created separately for different experiments or analyses.

4.2 Creating a HiCData object

In order to create a HiC dataset, you first need to have created an appropriate Fend object. You can create the HiCData object from a file containing fend indices and their observed counts, the format used with HiCPipe data (the 'mat' format), a text file containing pairs of chromosomes, coordinates, and strands (referred to from now on as a 'raw' file), or directly from mapped data in BAM format. If using raw coordinate data to create the dataset, the format should be:

```
chr1  coord1   strand1    chr2    coord2  strand2
```

where values are separated by tabs and strands are denoted by '+' or '-'. In addition to mapped reads, you need to provide a maximum insert size, i.e. the total acceptable length of a sequenced fragment as determined by the sum of each mapping location to its downstream RE cutsite.

To create the 5C dataset from a HiCPipe-compatible format, you can run the following commands:

```
import hifive
data = hifive.HiCData(out_filename, mode='w')
data.load_data_from_mat(fend_filename, data_filename)
data.save()
```

In this case, 'out_filename' specifies the location to save the `HiCData` object to and should end with the '.hdf5' extension. The 'fend_filename' value is the location of the appropriate `Fend` object. To maintain compatibility with HiCPipe-formatted 'mat' files, `HiFive` expects that fend and fragment numbering begin at index 1, not 0.

To create the 5C dataset from raw coordinate data, you can run the following commands:

```
import hifive
data = hifive.HiCData(out_filename, mode='w')
data.load_data_from_raw(fend_filename, [raw1.txt, raw2.txt])
data.save()
```

In this case, 'out_filename' specifies the location to save the `HiCData` object to and should end with the '.hdf5' extension. The 'fend_filename' value is the location of the appropriate `Fend` object. Multiple files containing paired-end mapped coordinates may be passed to the function as a list or, if only a single file is needed, it may be passed as a string.

In order to load data from a set of BAM files, a similar procedure is used:

```
import hifive
data = hifive.HiCData(out_filename, mode='w')
data.load_data_from_bam(fragment_filename, [bam_prefix1, bam_prefix2])
data.save()
```

In this case, the only difference is that prefices are passed instead of complete file names. The prefices should be the entire file path up until the strand specifier such that the two file names are created by appending either '_1.bam*' or '_2.bam*' to the prefix. The asterix denotes that all files with that name but different suffices are loaded. This allows files that are generated by multiple rounds of alignment that differ only by a suffix can be loaded without additional manipulation. Like the function for raw data, if only a single prefix is needed it may be passed as a string.

Note: The `HiCData` object can now be used by multiple analyses of this sample and does not need to be created separately for each one.

4.3 Creating a `HiC` analysis object

The HiC analysis object, `HiC`, contains links to a HiCData and `Fend` object, information about which fends to include in the analysis, model parameters, and learned model values. This is the standard way of working with HiC data in `HiFive` and this object will be used for learning the model, extracting portions of data, plotting, and downstream analysis.

To create a `HiC` object, you can use the following commands:

```
import hifive
hic = hifive.HiC(out_filename, 'w')
```

```
hic.load_data(data_filename)
hic.save()
```

In this case, 'out_filename' specifies the location to save the HiC object to and should end with the '.hdf5' extension. The 'data_filename' value is the location of the appropriate data object.

Warning: Becauase data and fragment data are stored in their own objects, each object keeps track of the location of its dependents through relative file names. This means that links between them will break if the relative pathway is changed.

4.4 Filter HiC fends

Prior to modeling the data, you need to filter out fends that have few valid reads mapped to them. HiFive uses an iterative filtering approach such that only when all fends satisfy a user-defined minimum number of valid interactions does the filtering process cease.

To filter fends, you can use the following commands:

```
import hifive
hic = hifive.HiC(hic_filename)
hic.filter_fends(mininteractions=25, maxdistance=5000000)
hic.save()
```

In this case, 'hic_filename' is a previously saved HiC analysis object. No value was passed to mode, since it defaults to 'r' for read. This loads the data from a previously created HiCData object. In order for changes to be kept to a FiveC object, it must be written to file using the save command. The 'maxdistance' argument specifies that only reads associated with interactions spanning that distance or less are counted for purposes of filtering fends.

4.5 Find HiC distance function

HiFive approximates the distance-signal relationship using a series of linear transitions between bin means of mean log interaction counts. Spanning from a user-defined minimum interaction distance up to the genome maximum interaction distance, the range is divided into equal-sized log distance bins. Values falling between bin midpoints are interpolated based on a linear transition between bins. To do an initial estimate of this function, you can use the following command:

```
hic.find_distance_means(numbins=90,
                        minsize=200,
                        maxsize=0,
                        smoothed=2,
                        corrected=False)
```

In this function call, the range of interaction sizes is being broken into 90 bins, 1 bin covering interactions <= 200 bp, and the other 89 spanning up to the maximum interaction distance with breaks evenly spaced in log space. The maximum of this range is set by 'maxsize', which can either be zero, as in this call, setting the maximum size equal to the longest interaction distance, or a positive integer value which would exclude any interaction distances greater than 'maxsize'. The 'smoothed' keyword specifies whether to apply a triangular smoothing to bin means after finding them. A value of zero would remain unsmoothed, one extends to using adjacent values, two would include values up to two steps away, and so on. The final argument, 'corrected', specifies whether to apply fend corrections to the interaction counts prior to calculating bin means.

Because this function involves scanning large amounts of data, it has been made to utilize MPI. To do so, you can use a scripts such as the following:

```
import hifive
from mpi4py import MPI

rank = MPI.COMM_WORLD.Get_rank()
hic = hifive.HiC(hic_filename)
hic.find_distance_means(numbins=90,
                        minsize=200,
                        maxsize=0,
                        smoothed=2,
                        corrected=False)
if rank == 0:
  hic.save()
```

4.6 Learn HiC normalization parameters

In order to learn the correction model for HiC data, `HiFive` uses two rounds of gradient descent, one with constant learning rate (the 'burn-in' phase) and the second with a linearly decreasing learning rate (the 'annealing' phase). In addition, `HiFive` can recalculate the distance function parameters periodically using the correction-adjusted interaction values. Finally, `HiFive` limits which interactions it uses to learn the model parameters to those that fall within a user-specified maximum interaction distance.

To learn HiC corrections using the modeling approach, you can use the following command:

```
hic.find_fend_corrections(display=100,
                          maxdistance=5000000,
                          learningrate=0.01,
                          burnin_iterations=5000,
                          annealing_iterations=10000,
                          recalculate_distance=2500)
```

In the above call, 'maxdistance' indicates that interctions spanning more than 5 Mb are excluded from calculations. Setting this to zero would include all unfiltered cis interactions. The 'recalculate_distance' parameters specifies how many iterations to wait before recalculating the distance parameters. The 'learningrate' specifies what percentage of the gradient to apply towards value updates. One last value passed to the function in 'display', which specifies how many iterations should pass before updating the display (via STDERR). This can also be set to zero to not display the progress.

Because of the large numbers of calculations involved in this function, it has been made to utilize MPI. To do so, you can use a scripts such as the following:

```
import hifive
from mpi4py import MPI

rank = MPI.COMM_WORLD.Get_rank()
hic = hifive.HiC(hic_filename)
hic.find_fend_corrections(display=100,
                          maxdistance=5000000,
                          learningrate=0.01,
                          burnin_iterations=5000,
                          annealing_iterations=10000,
                          recalculate_distance=2500)
if rank == 0:
  hic.save()
```

4.7 Approximate HiC normalization parameters

HiFive also offers an approximation approach for learning correction values. The primary differences to the correction model from the user's perspective are a single learning phase and a lack of learning rate. The approximation learning approach can still recalculate the distance function parameters periodically.

To learn HiC corrections using the approximation approach, you can use the following command:

```
hic.find_express_fend_corrections(iterations=1000,
                                  mindistance=0,
                                  usereads='cis',
                                  remove_distance=True,
                                  recalculate_distance=200)
```

In the above call, 'mindistance' is used to exclude interaction distances shorter that the passed value. If this results in the exclusion of any reads, fends are refiltered using either the value passed under the keyword 'mininteractions' or, if that is not specified, the value passed the last time fends were filtered. The 'usereads' argument allows the user to base the correction value approximation on 'cis' interactions, 'trans' interactions, or 'all'. Selecting 'trans' interactions will also result in a refiltering of fends to ensure that all of them are involved in sufficient interactions as described previously. The 'remove_distance' argument specifies whether to remove the distance-dependent portion of the signal prior to approximating correction values. For best results, this should set to true (its default value).

Although this function is much more computationally efficient, the recalculation of the distance function can take time and so has been made to utilize the MPI environment when available as follows:

```
import hifive
from mpi4py import MPI

rank = MPI.COMM_WORLD.Get_rank()
hic = hifive.HiC(hic_filename)
hic.find_express_fend_corrections(iterations=1000,
                                  mindistance=0,
                                  usereads='cis',
                                  remove_distance=True,
                                  recalculate_distance=200)
if rank == 0:
  hic.save()
```

4.8 Generating a heatmap

In order to immediately make use of data, HiFive allows you to pull data from a region and create a heatmap. The data can be returned unbinned, binned using a fixed-width bin size, or binned using boundaries passed by the user. There are several options for the format the data can be passed back in. Please refer to the cis_heatmap function for more details. There are also several options for transformations to the data. These are used to remove the distance-dependence signal, fend bias, both, or to return only the predicted signal. In this example, we'll get a portion of chromosome 1 binned into 10 Kb bins as follows:

```
heatmap = hic.cis_heatmap(chrom='1',
                          start=1000000
                          stop=3000000
                          binsize=10000,
                          arraytype='upper',
                          datatype='enrichment')
```

In the above call, All valid possible interactions were queried from chromosome 1 between 1000000 and 3000000. For valid interactions that had no observation, an expected value was still added to the bin. 'enrichment' specifies

to find the observed counts and expected counts, which includes the distance-dependence and fend bias values. The observed counts are in the first index of the last dimension of the returned array, the expected counts are in the second index of the last dimension. 'Upper' specifies a row-major upper triangle array (all values above the matrix diagonal flattened).

4.9 Plotting a heatmap

In order to visualize the heatmap we just produced, `HiFive` has several plotting functions that take different shaped arrays. The function called needs to match the array produced. In this case, we produced an upper array which is compatible with the `hifive.plotting.plot_upper_array()` function, so we'll use that as follows:

```
img = hifive.plotting.plot_upper_array(heatmap, symmetric_scaling=True)
img.save(out_fname)
```

In calling the function, we pass the heatmap and that would be sufficient. There are, however, additional options. For example, 'symmetric_scaling' specifies whether the color scale should be expanded to run from the minimum value to the maximum (False) or so that the maximum absolute value determine both upper and lower color bounds. The image returnd is a `PIL` image of type 'png'.

Note: The next thing on the todo list is write wrappers within the `FiveC` and `HiC` classes for running plotting through the analysis objects themselves.

CHAPTER 5

The Fragment class

class hifive.fragment.**Fragment** (*filename*, *mode='r'*)

This class handles restriction enzyme digest-generated fragment data for 5C experiments.

This class stores a list of chromosomes, a dictionary for converting from chromosome label to integer and back, fragment starts, stops, and chromosome number in an h5dict.

Note: This class is also available as hifive.Fragment

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns Fragment class object.

load_fragments (*filename*, *genome_name=None*, *re_name=None*, *regions=[]*, *minregionspacing=1000000*)

Parse and store fragment data from a bed for a 5C assay file into an h5dict.

Parameters

- **filename** (*str.*) – A file name to read restriction fragment data from. This should be a BED file containing fragment boundaries for all probed fragments and primer names that match those used for read mapping.
- **genome_name** (*str.*) – The name of the species and build. Optional.
- **re_name** (*str.*) – The name of the restriction enzyme used to produce the fragment set. Optional.
- **regions** (*list*) – User-defined partitioning of fragments into different regions. This argument should be a list of lists containing the chromosome, start, and stop coordinates for each region.
- **minregionspacing** (*int.*) – If 'regions' is not defined, this is used to parse regions by inserting breaks where fragments are spaced apart greater than this value.

Returns None

save()

Save fragment data to h5dict.

Returns None

CHAPTER 6

The Fend class

class hifive.fend.**Fend**(*filename*, *mode='r'*)

This class handles restriction enzyme digest-generated fragment data for HiC experiments.

This class stores a list of chromosomes, a dictionary for converting from chromosome label to integer and back, fragment starts, stops, and chromosome number in an h5dict.

Note: This class is also available as hifive.Fend

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns Fend class object

load_fends(*filename*, *genome_name=None*, *re_name=None*)

Parse and store fend data in h5dict.

Parameters

- **filename** (*str.*) – A file name to read restriction fragment data from. The file may be a 'mat' file compatible with HiCPipe, or a BED file containing RE fragment boundaries or cutsites.
- **genome_name** (*str.*) – The name of the species and build. Optional.
- **re_name** (*str.*) – The name of the restriction enzyme used to produce the fragment set. Optional.

Returns None

save()

Save fend data to h5dict.

Returns None

CHAPTER 7

The FiveCData class

class hifive.fivec_data.**FiveCData**(*filename*, *mode='r'*)

This class handles interaction count data for 5C experiments.

This class stores mapped paired-end reads, indexing them by fragment number, in an h5dict.

Note: This class is also available as hifive.FiveCData

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns FiveCData class object.

load_data_from_bam(*fragfilename*, *filelist*)

Read interaction counts from pairs of BAM files and place in h5dict.

Parameters

- **fragfilename** (*str.*) – This specifies the file name of the Fragment object to associate with the dataset.
- **filelist** (*list*) – A list containing all of the bam file prefices to be included in the dataset. All files containing each prefix will be loaded. If only one pair of files is needed, the prefix may be passed as a string.

Returns None

load_data_from_counts(*fragfilename*, *filelist*)

Read interaction counts from a text file(s) and place in h5dict.

Parameters

- **fragfilename** (*str.*) – This specifies the file name of the Fragment object to associate with the dataset.
- **filelist** (*list*) – A list containing all of the file names of counts text files to be included in the dataset. If only one file is needed, this may be passed as a string.

Returns None

save()

Save 5C interaction count data to h5dict.

Returns None

CHAPTER 8

The HiCData class

class hifive.hic_data.**HiCData** (*filename*, *mode='r'*)

This class handles interaction count data for HiC experiments.

This class stores mapped paired-end reads, indexing them by fragment-end (fend) number, in an h5dict.

Note: This class is also available as hifive.HiCData

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns HiCData class object.

export_to_mat (*outfilename*)

Write reads loaded in data object to text file in HiCPipe-compatible 'mat' format.

Parameters **outfilename** (*str.*) – Specifies the file to save data in.

Returns None

load_data_from_bam (*fendfilename*, *filelist*, *maxinsert*)

Read interaction counts from pairs of BAM-formatted alignment file(s) and place in h5dict.

Parameters

- **fendfilename** (*str.*) – This specifies the file name of the Fend object to associate with the dataset.
- **filelist** (*list*) – A list containing all of the bam file prefices to be included in the dataset. All files containing each prefix will be loaded. If only one pair of files is needed, the prefix may be passed as a string.
- **maxinsert** (*int.*) – A cutoff for filtering paired end reads whose total distance to their respective restriction sites exceeds this value.

Returns None

load_data_from_mat (*fendfilename*, *filename*, *maxinsert=0*)

Read interaction counts from a HiCPipe-compatible 'mat' text file and place in h5dict.

Parameters

- **fendfilename** (*str.*) – This specifies the file name of the `Fend` object to associate with the dataset.
- **filename** (*str.*) – File name of a 'mat' file containing fend pair and interaction count data.
- **maxinsert** (*int.*) – A cutoff for filtering paired end reads whose total distance to their respective restriction sites exceeds this value.

Returns None

load_data_from_raw (*fendfilename*, *filelist*, *maxinsert*)

Read interaction counts from a text file(s) and place in h5dict.

Files should contain both mapped ends of a read, one read per line, separated by tabs. Each line should be in the following format:

```
chromosome1    coordinate1  strand1   chromosome2    coordinate2  strand2
```

where strands are given by the characters '+' and '-'.

Parameters

- **fendfilename** (*str.*) – This specifies the file name of the `Fend` object to associate with the dataset.
- **filelist** (*list*) – A list containing all of the file names of mapped read text files to be included in the dataset. If only one file is needed, this may be passed as a string.
- **maxinsert** (*int.*) – A cutoff for filtering paired end reads whose total distance to their respective restriction sites exceeds this value.

Returns None

CHAPTER 9

The FiveC class

class hifive.fivec.**FiveC**(*filename*, *mode='r'*)

This is the class for handling 5C analysis.

This class relies on Fragment and FiveCData for genomic position and interaction count data. Use this class to perform filtering of fragments based on coverage, model fragment bias and distance dependence, and downstream analysis and manipulation. This includes binning of data, plotting of data, and statistical analysis.

Note: This class is also available as hifive.FiveC

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns FiveC class object.

cis_heatmap(*region*, *start=None*, *stop=None*, *startfrag=None*, *stopfrag=None*, *binsize=0*, *datatype='enrichment'*, *arraytype='full'*, *skipfiltered=False*, *returnmapping=False*, *dynamically_binned=False*, *minobservations=0*, *searchdistance=0*, *expansion_binsize=0*, *removefailed=False*)

Return a heatmap of cis data of the type and shape specified by the passed arguments.

This function returns a heatmap for a single region, bounded by either 'start' and 'stop' or 'startfend' and 'stopfend' ('start' and 'stop' take precedence). If neither is given, the complete region is included. The data in the array is determined by the 'datatype', being raw, fragment-corrected, distance-corrected, enrichment, or expected data. The array shape is given by 'arraytype' and can be compact (if unbinned), upper, or full. See fivec_binning for further explanation of 'datatype' and 'arraytype'. If using dynamic binning ('dynamically_binned' is set to True), 'minobservations', 'searchdistance', 'expansion_binsize', and 'removefailed' are used to control the dynamic binning process. Otherwise these arguments are ignored.

Parameters

- **region** (*int.*) – The index of the region to obtain data from.
- **start** (*int.*) – The smallest coordinate to include in the array, measured from fragment midpoints. If both 'start' and 'startfrag' are given, 'start' will override 'startfrag'. If unspecified, this will be set to the midpoint of the first fragment for 'region'. Optional.
- **stop** (*int.*) – The largest coordinate to include in the array, measured from fragment midpoints. If both 'stop' and 'stopfrag' are given, 'stop' will override 'stopfrag'. If unspecified, this will be set to the midpoint of the last fragment plus one for 'region'. Optional.

- **startfrag** (*int.*) – The first fragment to include in the array. If unspecified and 'start' is not given, this is set to the first fragment in 'region'. In cases where 'start' is specified and conflicts with 'startfrag', 'start' is given preference. Optional
- **stopfrag** (*str.*) – The first fragment not to include in the array. If unspecified and 'stop' is not given, this is set to the last fragment in 'region' plus one. In cases where 'stop' is specified and conflicts with 'stopfrag', 'stop' is given preference. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin. If 'binsize' is zero, unbinned data is returned.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact' (if unbinned), 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N and M are the number of forward and reverse probe fragments, respectively. 'full' returns a square, symmetric array of size N x N x 2 where N is the total number of fragments or bins. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2, where N is the total number of fragments or bins.
- **skipfiltered** (*bool.*) – If True, all interaction bins for filtered out fragments are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If True, a list containing the data array and either a 1d array containing fragment numbers included in the data array if the array is not compact or two 1d arrays containin fragment numbers for forward and reverse fragments if the array is compact is return. Otherwise only the data array is returned.
- **dynamically_binned** (*bool.*) – If True, return dynamically binned data.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **expansion_binsize** (*int.*) – The size of bins to use for data to pull from when expanding dynamic bins. If set to zero, unbinned data is used.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns Array in format requested with 'arraytype' containing data requested with 'datatype'. If returnmapping is True, a list is returned containined the requested data array and an array of associated positions (dependent on the binning options selected).

filter_fragments(*mininteractions=20*)

Iterate over the dataset and remove fragments that do not have 'minobservations' using only unfiltered fragments.

In order to create a set of fragments that all have the necessary number of interactions, after each round of filtering, fragment interactions are retallied using only interactions that have unfiltered fragments at both

ends.

Parameters **mininteractions** (*int.*) – The required number of interactions for keeping a fragment in analysis.

Returns None

find_distance_parameters()

Regress log counts versus inter-fragment distances to find slope and intercept values and then find the standard deviation of corrected counts.

Returns None

find_express_fragment_corrections(*iterations=1000*, *remove_distance=False*, *recalculate_distance=100*)

Using iterative approximation, learn correction values for each valid fragment.

Parameters

- **iterations** (*int.*) – The number of iterations to use for learning fragment corrections.
- **remove_distance** (*bool.*) – Specifies whether the estimated distance-dependent portion of the signal is removed prior to learning fragment corrections.
- **recalculate_distance** (*int.*) – Number of iterations that should pass before recalculating the distance bin means to account for the updated fragment corrections. If set to zero, no recalculation is performed.

Returns None

find_fragment_corrections(*maxdistance=0*, *burnin_iterations=5000*, *annealing_iterations=10000*, *learningrate=0.01*, *recalculate_distance=100*, *display=10*)

Using gradient descent, learn correction values for each valid fragment based on a Log-Normal distribution of observations.

Parameters

- **maxdistance** (*int.*) – The maximum inter-fragment distance to be included in modeling.
- **burnin_iterations** (*int.*) – The number of iterations to use with constant learning rate in gradient descent for learning fragment corrections.
- **annealing_iterations** (*int.*) – The number of iterations to use with a linearly-decreasing learning rate in gradient descent for learning fragment corrections.
- **learningrate** (*float*) – The gradient scaling factor for parameter updates.
- **recalculate_distance** (*int.*) – Number of iterations that should pass before recalculating the distance function parameters to account for the updated fragment corrections. If set to zero, no recalculation is performed.
- **display** (*int.*) – Specifies how many iterations between when cost is calculated and displayed as model is learned. If 'display' is zero, the cost is not calculated of displayed.

Returns None

find_trans_mean()

Calculate the mean signal across all valid fragment-pair trans (inter-region) interactions.

Returns None

load()
Load analysis parameters from h5dict specified at object creation and open h5dicts for associated FiveCData and Fragment objects.

Any call of this function will overwrite current object data with values from the last save() call.

Returns None

load_data(*filename*)
Load fragment-pair counts and fragment object from FiveCData object.

Parameters **filename** (*str.*) – Specifies the file name of the FiveCData object to associate with this analysis.

Returns None

save()
Save analysis parameters to h5dict.

Returns None

trans_heatmap(*region1*, *region2*, *start1=0*, *stop1=None*, *startfrag1=None*, *stopfrag1=None*, *start2=0*, *stop2=None*, *startfrag2=None*, *stopfrag2=None*, *binsize=1000000*, *datatype='enrichment'*, *arraytype='full'*, *returnmapping=False*, *dynamically_binned=False*, *minobservations=0*, *searchdistance=0*, *expansion_binsize=0*, *removefailed=False*)
Return a heatmap of trans data of the type and shape specified by the passed arguments.

This function returns a heatmap for trans interactions between two regions, bounded by either 'start1', 'stop1', 'start2' and 'stop2' or 'startfrag1', 'stopfrag1', 'startfrag2', and 'stopfrag2' ('start' and 'stop' take precedence). The data in the array is determined by the 'datatype', being raw, fragment-corrected, distance-corrected, enrichment, or expected data. The array shape is always rectangular but can be either compact (which returns two arrays) or full. See fivec_binning for further explanation of 'datatype' and 'arraytype'. If using dynamic binning ('dynamically_binned' is set to True), 'minobservations', 'searchdistance', 'expansion_binsize', and 'removefailed' are used to control the dynamic binning process. Otherwise these arguments are ignored.

Parameters

- **region1** (*int.*) – The index of the first region to obtain data from.
- **region2** (*int.*) – The index of the second region to obtain data from.
- **start1** (*int.*) – The coordinate at the beginning of the smallest bin from 'region1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfrag1' mid. If there is a conflict between 'start1' and 'startfrag1', 'start1' is given preference. Optional.
- **stop1** (*int.*) – The largest coordinate to include in the array from 'region1', measured from fragment midpoints. If both 'stop1' and 'stopfrag1' are given, 'stop1' will override 'stopfrag1'. 'stop1' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag1** (*int.*) – The first fragment from 'region1' to include in the array. If unspecified and 'start1' is not given, this is set to the first valid fend in 'region1'. In cases where 'start1' is specified and conflicts with 'startfrag1', 'start1' is given preference. Optional.
- **stopfrag1** (*int.*) – The first fragment not to include in the array from 'region1'. If unspecified and 'stop1' is not given, this is set to the last valid fragment in 'region1' + 1. In cases where 'stop1' is specified and conflicts with 'stopfrag1', 'stop1' is given preference. Optional.

- **start1** – The coordinate at the beginning of the smallest bin from 'region1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfrag1' mid. If there is a conflict between 'start1' and 'startfrag1', 'start1' is given preference. Optional.
- **stop2** (*int.*) – The largest coordinate to include in the array from 'region2', measured from fragment midpoints. If both 'stop2' and 'stopfrag2' are given, 'stop2' will override 'stopfrag2'. 'stop2' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag2** (*int.*) – The first fragment from 'region2' to include in the array. If unspecified and 'start2' is not given, this is set to the first valid fend in 'region2'. In cases where 'start2' is specified and conflicts with 'startfrag2', 'start2' is given preference. Optional.
- **stopfrag2** (*int.*) – The first fragment not to include in the array from 'region2'. If unspecified and 'stop2' is not given, this is set to the last valid fragment in 'region2' + 2. In cases where 'stop2' is specified and conflicts with 'stopfrag2', 'stop2' is given preference. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are aways in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, non-filtered bins return value of 1. Expected values are returned for 'distance', 'fragment', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact' (if unbinned) and 'full'. 'compact' means data are arranged in a N x M x 2 array where N and M are the number of forward and reverse probe fragments, respectively. Two arrays will be returned for this format, the first with forward probe fragments from region1 and reverse probe fragments from region2. The second is the compliment of the first. 'full' returns a square, symmetric array of size N x N x 2 where N is the total number of fragments or bins.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and mapping information is returned. Otherwise only a data array(s) is returned.
- **dynamically_binned** (*bool.*) – If 'True', return dynamically binned data.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **expansion_binsize** (*int.*) – The size of bins to use for data to pull from when expanding dynamic bins. If set to zero, unbinned data is used.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns Array in format requested with 'arraytype' containing inter-region data requested with 'datatype'. If 'returnmapping' is True, a list is returned with mapping information. If 'arraytype' is 'full', a single data array and two 1d arrays of fragments corresponding to rows and columns, respectively is returned. If 'arraytype' is 'compact', two data arrays are returned

(forward1 by reverse2 and forward2 by reverse1) along with forward and reverse fragment positions for each array for a total of 5 arrays.

write_heatmap_dict (*filename*, *binsize*, *includetrans=True*, *remove_distance=False*, *arraytype='full'*, *regions=[]*)

Create an h5dict file containing binned interaction arrays, bin positions, and an index of included regions.

Parameters

- **filename** (*str.*) – Location to write h5dict object to.
- **binsize** (*int.*) – Size of bins for interaction arrays. If "binsize" is zero, fragment interactions are returned without binning.
- **includetrans** (*bool.*) – Indicates whether trans interaction arrays should be calculated and saved.
- **remove_distance** (*bool.*) – If 'True', the expected value is calculated including the expected distance mean. Otherwise, only fragment corrections are used.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact' and 'full'. 'compact' means data are arranged in a N x M x 2 array where N is the number of bins, M is the maximum number of steps between included bin pairs, and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2.
- **regions** (*list.*) – If given, indicates which regions should be included. If left empty, all regions are included.

Returns None

CHAPTER 10

The HiC class

class hifive.hic.**HiC**(*filename*, *mode='r'*)

This is the class for handling HiC analysis.

This class relies on Fend and HiCData for genomic position and interaction count data. Use this class to perform filtering of fends based on coverage, model fend bias and distance dependence, and downstream analysis and manipulation. This includes binning of data, plotting of data, modeling of data, and statistical analysis.

Note: This class is also available as hifive.HiC

When initialized, this class creates an h5dict in which to store all data associated with this object.

Parameters

- **filename** (*str.*) – The file name of the h5dict. This should end with the suffix '.hdf5'
- **mode** (*str.*) – The mode to open the h5dict with. This should be 'w' for creating or overwriting an h5dict with name given in filename.

Returns HiC class object.

cis_heatmap(*chrom*, *start=0*, *stop=None*, *startfend=None*, *stopfend=None*, *binsize=0*, *binbounds=None*, *datatype='enrichment'*, *arraytype='compact'*, *maxdistance=0*, *skipfiltered=False*, *returnmapping=False*, *dynamically_binned=False*, *minobservations=0*, *searchdistance=0*, *expansion_binsize=0*, *removefailed=False*)

Return a heatmap of cis data of the type and shape specified by the passed arguments.

This function returns a heatmap for a single chromosome region, bounded by either 'start' and 'stop' or 'startfend' and 'stopfend' ('start' and 'stop' take precedence), or if given, the outer coordinates of the array passed by 'binbounds'. If none of these are specified, data for the complete chromosome is used. The data in the array is determined by the 'datatype', being raw, fend-corrected, distance-corrected, enrichment, or expected data. The array shape is given by 'arraytype' and can be compact, upper, or full. See hic_binning for further explanation of 'datatype' and 'arraytype'. The returned data will include interactions ranging from zero to 'maxdistance' apart. If maxdistance is zero, all interactions within the requested bounds are returned. If using dynamic binning ('dynamically_binned' is set to True), 'minobservations', 'searchdistance', 'expansion_binsize', and 'removefailed' are used to control the dynamic binning process. Otherwise these arguments are ignored.

Parameters

- **chrom** (*str.*) – The name of a chromosome to obtain data from.
- **start** (*int.*) – The smallest coordinate to include in the array, measured from fend midpoints. If both 'start' and 'startfend' are given, 'start' will override 'startfend'. If unspecified, this will be set to the midpoint of the first fend for 'chrom'. Optional.

- **stop** (*int.*) – The largest coordinate to include in the array, measured from fend midpoints. If both 'stop' and 'stopfend' are given, 'stop' will override 'stopfend'. If unspecified, this will be set to the midpoint of the last fend plus one for 'chrom'. Optional.
- **startfend** (*int.*) – The first fend to include in the array. If unspecified and 'start' is not given, this is set to the first fend in 'chrom'. In cases where 'start' is specified and conflicts with 'startfend', 'start' is given preference. Optional
- **stopfend** (*str.*) – The first fend not to include in the array. If unspecified and 'stop' is not given, this is set to the last fend in 'chrom' plus one. In cases where 'stop' is specified and conflicts with 'stopfend', 'stop' is given preference. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin. If 'binsize' is zero, unbinned data is returned. If 'binbounds' is not None, this value is ignored.
- **binbounds** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins. Any fend not falling in a bin is ignored. Optional.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact', 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N is the number of fends or bins, M is the maximum number of steps between included fend pairs or bin pairs and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2.
- **maxdistance** (*str.*) – This specifies the maximum coordinate distance between bins that will be included in the array. If set to zero, all distances are included.
- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fends are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 1d array containing fend numbers included in the data array if unbinned or a 2d array of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates if binned is return. Otherwise only the data array is returned.
- **dynamically_binned** (*bool.*) – If 'True', return dynamically binned data.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **expansion_binsize** (*int.*) – The size of bins to use for data to pull from when expanding dynamic bins. If set to zero, unbinned data is used.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns Array in format requested with 'arraytype' containing data requested with 'datatype'. If returnmapping is True, a list is returned containined the requested data array and an array of associated positions (dependent on the binning options selected).

filter_fends(*mininteractions=10*, *maxdistance=0*)

Iterate over the dataset and remove fends that do not have 'minobservations' within 'maxdistance' of themselves using only unfiltered fends.

In order to create a set of fends that all have the necessary number of interactions, after each round of filtering, fend interactions are retallied using only interactions that have unfiltered fends at both ends.

Parameters

- **mininteractions** (*int.*) – The required number of interactions for keeping a fend in analysis.
- **maxdistance** (*int.*) – The maximum inter-fend distance used to count fend interactions. A value of 0 indicates all cis-data should be used.

Returns None

find_distance_means(*numbins=90*, *minsize=200*, *maxsize=0*, *smoothed=0*, *startfend=0*, *stopfend=None*, *corrected=False*)

Count reads and possible interactions from valid fend pairs in each distance bin to find mean bin signals. This function is MPI compatible.

This partitions the range of interaction distances (measured from mipoints of the involved fends) from the 'minsize' to 'maxsize' into a number of partitions equal to 'numbins'. The first bin contains all distances less than or equal to 'minsize'. The remaining bins are defined such that their log ranges are equal to one another. The curve defined by the mean interaction value of each bin can be smoothed using a triangular smoothing operation.

Parameters

- **numbins** (*int.*) – The number of bins to divide the distance range into. The first bin extends from zero to 'minsize', while the remaining bins are divided into evenly-spaced log-sized bins from 'minsize' to 'maxsize' or the maximum inter-fend distance, whichever is greater.
- **minsize** (*int.*) – The upper size limit of the smallest distance bin.
- **maxsize** (*int.*) – If this value is larger than the largest included chromosome, it will extend bins out to maxsize. If this value is smaller, it is ignored.
- **smoothed** (*int.*) – Indicates the degree of smoothing to the distance curve for noise reduction. A value of less than 1 indicates no smoothing.
- **startfend** (*int.*) – The first fend to include interactions from in calculating bin means.
- **stopfend** (*int.*) – The first fend to exclude interactions from in calculating bin means. If stopfend is None, there is no upper bound on included fends.
- **corrected** (*bool.*) – If True, correction values are applied to counts prior to summing.

Returns None

find_express_fend_corrections(*iterations=100*, *mindistance=0*, *remove_distance=True*, *usereads='cis'*, *recalculate_distance=0*, *mininteractions=None*)

Using iterative approximation, learn correction values for each valid fend. This function is MPI compatible.

Parameters

- **iterations** (*int.*) – The number of iterations to use for learning fend corrections.
- **mindistance** (*int.*) – This is the minimum distance between fend midpoints needed to be included in the analysis. All possible and observed interactions with a distance shorter than this are ignored. If 'usereads' is set to 'trans', this value is ignored.
- **remove_distance** (*bool.*) – Specifies whether the estimated distance-dependent portion of the signal is removed prior to learning fend corrections.
- **usereads** (*str.*) – Specifies which set of interactions to use, 'cis', 'trans', or 'all'.
- **recalculate_distance** (*int.*) – Number of iterations that should pass before recalculating the distance bin means to account for the updated fend corrections. If set to zero, no recalculation is performed.
- **mininteractions** (*int.*) – If a non-zero 'mindistance' is specified or only 'trans' interactions are used, fend filtering will be performed again to ensure that the data being used is sufficient for analyzed fends. This parameter may specify how many interactions are needed for valid fends. If not given, the value used for the last call to `filter_fends()` is used or, barring that, one.

Returns None

find_fend_corrections(*maxdistance=0*, *burnin_iterations=10000*, *annealing_iterations=10000*, *learningrate=0.01*, *recalculate_distance=0*, *display=0*)

Using gradient descent, learn correction values for each valid fend based on a Poisson distribution of observations. This function is MPI compatible.

Parameters

- **maxdistance** (*int.*) – The maximum inter-fend distance to be included in modeling.
- **burnin_iterations** (*int.*) – The number of iterations to use with constant learning rate in gradient descent for learning fend corrections.
- **annealing_iterations** (*int.*) – The number of iterations to use with a linearly-decreasing learning rate in gradient descent for learning fend corrections.
- **learningrate** (*float*) – The gradient scaling factor for parameter updates.
- **recalculate_distance** (*int.*) – Number of iterations that should pass before recalculating the distance bin means to account for the updated fend corrections. If set to zero, no recalculation is performed.
- **display** (*int.*) – Specifies how many iterations between when cost is calculated and displayed as model is learned. If 'display' is zero, the cost is not calculated of displayed.

Returns None

find_trans_mean()

Calculate the mean signal across all valid fend-pair trans interactions.

Returns None

learn_fend_3D_model(*chrom*, *minobservations=10*)

Learn coordinates for a 3D model of data using an approximate PCA dimensional reduction.

This function makes use of the `mlpy` function `PCAFast()` to reduce the data to a set of three coordinates per fend. Cis data for all unfiltered fends for the specified chromosome are dynamically binned to yield a complete distance matrix. The diagonal is set equal to the highest valid enrichment value after dynamic binning. This N x N matrix is passed to `PCAFast()` and reduced to an N x 3 matrix.

Parameters

- **chrom** (*str.*) – The chromosome to learn the model for.

- **minobservations** (*int.*) – The minimum number of observed reads needed to cease bin expansion in the dynamic binning phase.

Returns Array containing a row for each valid fend and columns containing X coordinate, Y coordinate, Z coordinate, and sequence coordinate (fend midpoint).

load()

Load analysis parameters from h5dict specified at object creation and open h5dicts for associated `HiCData` and `Fend` objects.

Any call of this function will overwrite current object data with values from the last `save()` call.

Returns None

load_data(*filename*)

Load fend-pair counts and fend object from `HiCData` object.

Parameters **filename** (*str.*) – Specifies the file name of the `HiCData` object to associate with this analysis.

Returns None

save()

Save analysis parameters to h5dict.

Returns None

trans_heatmap(*chrom1*, *chrom2*, *start1=0*, *stop1=None*, *startfend1=None*, *stopfend1=None*, *binbounds1=None*, *start2=0*, *stop2=None*, *startfend2=None*, *stopfend2=None*, *binbounds2=None*, *binsize=1000000*, *datatype='enrichment'*, *returnmapping=False*, *dynamically_binned=False*, *minobservations=0*, *searchdistance=0*, *expansion_binsize=0*, *removefailed=False*)

Return a heatmap of trans data of the type and shape specified by the passed arguments.

This function returns a heatmap for trans interactions between two chromosomes within a region, bounded by either 'start1', 'stop1', 'start2' and 'stop2' or 'startfend1', 'stopfend1', 'startfend2', and 'stopfend2' ('start' and 'stop' take precedence), or if given, the outer coordinates of the arrays passed by 'binbounds1' and 'binbounds2'. The data in the array is determined by the 'datatype', being raw, fend-corrected, distance-corrected, enrichment, or expected data. The array shape is always rectangular. See `hic_binning` for further explanation of 'datatype'. If using dynamic binning ('dynamically_binned' is set to True), 'minobservations', 'searchdistance', 'expansion_binsize', and 'removefailed' are used to control the dynamic binning process. Otherwise these arguments are ignored.

Parameters

- **chrom1** (*str.*) – The name of the first chromosome to obtain data from.
- **chrom2** (*str.*) – The name of the second chromosome to obtain data from.
- **start1** (*int.*) – The coordinate at the beginning of the smallest bin from 'chrom1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfend1' mid. If there is a conflict between 'start1' and 'startfend1', 'start1' is given preference. Optional.
- **stop1** (*int.*) – The largest coordinate to include in the array from 'chrom1', measured from fend midpoints. If both 'stop1' and 'stopfend1' are given, 'stop1' will override 'stopfend1'. 'stop1' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfend1** (*int.*) – The first fend from 'chrom1' to include in the array. If unspecified and 'start1' is not given, this is set to the first valid fend in 'chrom1'. In cases where 'start1' is specified and conflicts with 'startfend1', 'start1' is given preference. Optional

- **stopfend1** – The first fend not to include in the array from 'chrom1'. If unspecified and 'stop1' is not given, this is set to the last valid fend in 'chrom1' + 1. In cases where 'stop1' is specified and conflicts with 'stopfend1', 'stop1' is given preference. Optional.
- **binbounds1** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins to use for partitioning 'chrom1'. Any fend not falling in a bin is ignored.
- **start2** (*int.*) – The coordinate at the beginning of the smallest bin from 'chrom2'. If unspecified, 'start2' will be the first multiple of 'binsize' below the 'startfend2' mid. If there is a conflict between 'start2' and 'startfend2', 'start2' is given preference. Optional.
- **stop2** (*int.*) – The largest coordinate to include in the array from 'chrom2', measured from fend midpoints. If both 'stop2' and 'stopfend2' are given, 'stop2' will override 'stopfend2'. 'stop2' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfend2** (*int.*) – The first fend from 'chrom2' to include in the array. If unspecified and 'start2' is not given, this is set to the first valid fend in 'chrom2'. In cases where 'start2' is specified and conflicts with 'startfend2', 'start2' is given preference. Optional
- **stopfend2** (*str.*) – The first fend not to include in the array from 'chrom2'. If unspecified and 'stop2' is not given, this is set to the last valid fend in 'chrom2' + 1. In cases where 'stop2' is specified and conflicts with 'stopfend2', 'stop1' is given preference. Optional.
- **binbounds2** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins to use for partitioning 'chrom2'. Any fend not falling in a bin is ignored.
- **binsize** (*int.*) – This is the coordinate width of each bin. If binbounds is not None, this value is ignored.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and two 2d arrays of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates for the first and second chromosomes is returned. Otherwise only the data array is returned.
- **dynamically_binned** (*bool.*) – If 'True', return dynamically binned data.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **expansion_binsize** (*int.*) – The size of bins to use for data to pull from when expanding dynamic bins. If set to zero, unbinned data is used.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns Array in format requested with 'arraytype' containing data requested with 'datatype'. If returnmapping is True, a list is returned containined the requested data array and an array

of associated positions (dependent on the binning options selected).

write_heatmap_dict (*filename*, *binsize*, *includetrans=True*, *remove_distance=False*, *chroms=[]*)
Create an h5dict file containing binned interaction arrays, bin positions, and an index of included chromosomes. This function is MPI compatible.

Parameters

- **filename** (*str.*) – Location to write h5dict object to.
- **binsize** (*int.*) – Size of bins for interaction arrays.
- **includetrans** (*bool.*) – Indicates whether trans interaction arrays should be calculated and saved.
- **remove_distance** (*bool.*) – If 'True', the expected value is calculated including the expected distance mean. Otherwise, only fend corrections are used.
- **chroms** (*list*) – A list of chromosome names indicating which chromosomes should be included. If left empty, all chromosomes are included. Optional.

Returns None

CHAPTER 11

The BI class

class hifive.bi.**BI**(*width=10000*, *window=1000000*, *height=0*, *mincount=10*)

This class uses a HiC or FiveC object to generate BI scores. These scores can be saved to an h5dict. Once generated or loaded, BI scores can be used to find structural boundaries, create feature-centered BI profiles, or bound-centered feature profiles.

Note: This class is also available as hifive.BI

The BI score is calculated by finding the absolute difference between a set of matched bins; the first set contains interactions between sequences falling within a range given by 'window' up- and downstream of the boundary point and binned into 'height'-sized bins and the group of sequences falling between zero and 'width' base pairs upstream of the boundary point; the second set contains interactions between sequences falling within a range given by 'window' up- and downstream of the boundary point and binned into 'height'-sized bins and the group of sequences falling between zero and 'width' base pairs downstream of the boundary point.

Parameters

- **width** (*int.*) – A range, in base-pairs, specifying how large a neighborhood around a boundary point is used for binning in calculating the BI.
- **height** (*int.*) – The size to bin fragments or fends into that fall within the range specified by 'window' of the boundary point and interacting with the fragments or fends follwing within the 'width' range around the boundary point. If set to zero, fragments or fends are left unbinned in this dimension for BI calculations.
- **window** (*int.*) – A range, in base-pairs, specifying how far away from a boundary point interactions are included in calculating the BI.
- **mincount** (*int.*) – The minimum number of valid pairs of bins needed to calculate the boundary index for a given location.

Returns BI class object

find_bi_bounds(*cutoff=2.0*, *chroms=*[ ])

Find BI scores that are higher than adjacent scores and are more than the cutoff * standard deviation above the minimum adjacent trough score.

Parameters

- **cutoff** (*float*) – Minimum height ratio between peak and lower adjacent trough.
- **chroms** (*list*) – If specified, find bounds for only chromosomes in this list. Otherwise use all chromosomes present in the BI object.

Returns Array containing a row for each valid BI score and columns contianing chromosome index, sequence coordinate, and BI score.

find_bi_from_fivec (*fivec*, *datatype='fragment'*, *regions=[]*)
Calculate the boundary index for 5C data.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fend and count data.
- **regions** (*list*) – A list of region numbers to limit BI calculations to. If empty, all regions in 'fivec' are used.

Returns None

find_bi_from_hic (*hic*, *datatype='fend'*, *chroms=[]*)
Calculate the boundary index from HiC data. This function is MPI compatible.

Parameters

- **hic** (HiC) – A HiC class object containing fend and count data.
- **chroms** (*list*) – A list of chromosome names to limit BI calculations to. If empty, all chromosomes in 'hic' are used.

Returns None

find_bi_profile (*coordinates*, *binsize=2000*, *numbins=51*)
Find the mean BI signal in a number of bins centered around a set of user-specified positions.

Parameters

- **coordinates** (*dict.*) – A dictionary keyed with chromosome names and containing 1d arrays of coordinates for each chromosome.
- **binsize** (*int.*) – Width, in base pairs, of each bin to aggregate signal in.
- **numbins** (*int.*) – The number of bins, centered around each coordinate, to aggregate signal in.

Returns An array of mean aggregate BI signal centered around 'coordinates'.

load (*filename*)
Load BI data and parameters from an h5dict.

Parameters **filename** (*str.*) – This specifies the file name of the BI object to load from.

Returns None

plot_bi (*width*, *height*, *chromosome*, *start=None*, *stop=None*)
Plot a line representation of the BI score for a specified region.

Parameters

- **width** (*float*) – The width of the image to return in whatever units Pyx is using.
- **height** (*float*) – The height of the image to return in whatever units Pyx is using.
- **chromosome** (*str.*) – The name of the chromosome to pull BI information from.
- **start** (*int.*) – The coordinates corresponding to the left edge of the image. If not specified, the first BI midpoint is used.
- **stop** (*int.*) – The coordinates corresponding to the right edge of the image. If not specified, the first BI midpoint is used.

Returns Pyx format canvas with line plot of bi scores.

save (*filename*)
Save BI data and parameters to an h5dict.

Parameters **filename** (*str.*) – This specifies the file name to which to save the BI object.

Returns None

smooth_bi(*width*)

Using a Gaussian weighting approach, smooth BI values. Weights extend out to 2.5 standard deviatons. This function is MPI compatible.

Parameters **width** (*int.*) – The distance, in base pairs, equivalent to one standard deviation for the Gaussian weights.

Returns None

CHAPTER 12

The fivec_binning module

This is a module contains scripts for generating compact, upper-triangle and full matrices of 5C interaction data.

12.1 Concepts

Data can either be arranged in compact, complete, or flattened (row-major) upper-triangle arrays. Compact arrays are N x M, where N is the number of forward probe fragments and M is the number of reverse probe fragments. Data can be raw, fragment-corrected, distance-dependence removed, or enrichment values. Arrays are 3-dimensional with observed values in the first layer of d3, expected values in the second layer of d3. The exception to this is upper-triangle arrays, which are 2d, dividing observed and expected along the second axis.

12.2 API documentation

hifive.fivec_binning.**bin_cis_array**(*fivec*, *unbinned*, *fragments*, *start=None*, *stop=None*, *binsize=10000*, *binbounds=None*, *arraytype='full'*, *returnmapping=False*)

Create an array of format 'arraytype' and fill 'binsize' bins or bins defined by 'binbounds' with data provided in 'unbinned'.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **unbinned** (*numpy array*) – A full or upper array containing data to be binned. Array format will be determined from the number of dimensions.
- **fragments** (*numpy array*) – A 1d integer array indicating which position corresponds to which fragment in the 'unbinned' array.
- **start** (*int.*) – The coordinate at the beginning of the smallest bin. If unspecified, 'start' will be the first multiple of 'binsize' below the first mid from 'fragments'. If 'binbounds' is given, 'start' is ignored. Optional.
- **stop** (*int.*) – The coordinate at the end of the last bin. If unspecified, 'stop' will be the first multiple of 'binsize' above the last mid from 'fragments'. If needed, 'stop' is adjusted upward to create a complete last bin. If 'binbounds' is given, 'stop' is ignored. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin. This is ignored if 'binbounds' is given.

- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'full' and 'upper'. 'full' returns a square, symmetric array of size N x N x 2. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 2d array of N x 4 containing the first fragment and last fragment plus one included in each bin and first and last coordinates is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing binned data requested with 'datatype' pulled from 'unbinned'.

hifive.fivec_binning.**bin_cis_signal**(*fivec*, *region*, *start=None*, *stop=None*, *startfrag=None*, *stopfrag=None*, *binsize=10000*, *datatype='enrichment'*, *arraytype='full'*, *returnmapping=False*)

Create an array of format 'arraytype' and fill 'binsize' bins with data requested in 'datatype'.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **region** (*int.*) – The index of the region to pull data from.
- **start** (*int.*) – The coordinate at the beginning of the smallest bin. If unspecified, 'start' will be the first multiple of 'binsize' below the 'startfend' mid. If there is a conflict between 'start' and 'startfrag', 'start' is given preference. Optional.
- **stop** (*int.*) – The largest coordinate to include in the array, measured from fend midpoints. If both 'stop' and 'stopfrag' are given, 'stop' will override 'stopfrag'. Optional.
- **startfrag** (*int.*) – The first fragment to include in the array. If unspecified and 'start' is not given, this is set to the first valid fragment in 'region'. In cases where 'start' is specified and conflicts with 'startfrag', 'start' is given preference. Optional.
- **stopfrag** (*int.*) – The first fragment not to include in the array. If unspecified and 'stop' is not given, this is set to the last valid fragment in 'region' + 1. In cases where 'stop' is specified and conflicts with 'stopfrad', 'stop' is given preference. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are aways in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, non-filtered expected bins return value of 1. Expected values are returned for 'distance', 'fragment', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'full' and 'upper'. 'full' returns a square, symmetric array of size N x N x 2 where N is the total number of fragments. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2, where N is the total number of fragments.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 2d array of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing binned data requested with 'datatype'.

hifive.fivec_binning.**bin_trans_signal**(*fivec*, *region1*, *region2*, *start1=None*, *stop1=None*, *startfrag1=None*, *stopfrag1=None*, *start2=None*, *stop2=None*, *startfrag2=None*, *stopfrag2=None*, *binsize=1000000*, *datatype='enrichment'*, *returnmapping=False*)

Create an array and fill 'binsize' bins with trans (inter-region) data requested in 'datatype'.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **region1** (*int.*) – The index of the first region to pull data from.
- **region2** (*int.*) – The index of the second region to pull data from.
- **start1** (*int.*) – The coordinate at the beginning of the smallest bin from 'region1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfrag1' mid. If there is a conflict between 'start1' and 'startfrag1', 'start1' is given preference. Optional.
- **stop1** (*int.*) – The largest coordinate to include in the array from 'region1', measured from fragment midpoints. If both 'stop1' and 'stopfrag1' are given, 'stop1' will override 'stopfrag1'. 'stop1' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag1** (*int.*) – The first fragment from 'region1' to include in the array. If unspecified and 'start1' is not given, this is set to the first valid fend in 'region1'. In cases where 'start1' is specified and conflicts with 'startfrag1', 'start1' is given preference. Optional.
- **stopfrag1** (*int.*) – The first fragment not to include in the array from 'region1'. If unspecified and 'stop1' is not given, this is set to the last valid fragment in 'region1' + 1. In cases where 'stop1' is specified and conflicts with 'stopfrag1', 'stop1' is given preference. Optional.
- **start2** (*int.*) – The coordinate at the beginning of the smallest bin from 'region2'. If unspecified, 'start2' will be the first multiple of 'binsize' below the 'startfrag2' mid. If there is a conflict between 'start2' and 'startfrag2', 'start2' is given preference. Optional.
- **stop2** (*int.*) – The largest coordinate to include in the array from 'region2', measured from fragment midpoints. If both 'stop2' and 'stopfrag2' are given, 'stop2' will override 'stopfrag2'. 'stop2' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag2** (*int.*) – The first fragment from 'region2' to include in the array. If unspecified and 'start2' is not given, this is set to the first valid fend in 'region2'. In cases where 'start2' is specified and conflicts with 'startfrag2', 'start2' is given preference. Optional.
- **stopfrag2** (*int.*) – The first fragment not to include in the array from 'region2'. If unspecified and 'stop2' is not given, this is set to the last valid fragment in 'region2' + 2. In cases where 'stop2' is specified and conflicts with 'stopfrag2', 'stop2' is given preference. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are aways in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, non-filtered bins return value of 1. Expected values are returned for 'distance', 'fragment', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and two 2d arrays of N x 4 containing the first fragment and last fragment plus one included in each bin and first

and last coordinates for 'region1' and 'region2' is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing binned inter-region data requested with 'datatype'.

hifive.fivec_binning.**dynamically_bin_cis_array**(*unbinned*, *unbinnedpositions*, *binned*, *binbounds*, *minobservations=50*, *searchdistance=0*, *removefailed=True*)

Expand bins in 'binned' to include additional data provided in 'unbinned' as necessary to meet 'minobservations', or 'searchdistance' criteria.

Parameters

- **unbinned** (*numpy array*) – A full or upper array containing data to be binned. Array format will be determined from the number of dimensions.
- **unbinnedpositions** (*numpy array*) – A 1d integer array indicating the mid-point of each bin in 'unbinned' array.
- **binned** (*numpy array*) – A full or upper array containing binned data to be dynamically binned. Array format will be determined from the number of dimensions. Data in this array will be altered by this function.
- **binbounds** (*numpy array*) – A N x 2 integer array indicating the start and end position of each of N bins in 'binned' array.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns None

hifive.fivec_binning.**dynamically_bin_trans_array**(*unbinned*, *unbinnedpositions1*, *unbinnedpositions2*, *binned*, *binbounds1*, *binbounds2*, *minobservations=50*, *searchdistance=0*, *removefailed=True*)

Expand bins in 'binned' to include additional data provided in 'unbinned' as necessary to meet 'minobservations', or 'searchdistance' criteria.

Parameters

- **unbinned** (*numpy array*) – A full array containing data to be binned.
- **unbinnedpositions1** (*numpy array*) – A 1d integer array indicating the mid-point of each bin in 'unbinned' array along the first axis.
- **unbinnedpositions2** (*numpy array*) – A 1d integer array indicating the mid-point of each bin in 'unbinned' array along the second axis.
- **binned** (*numpy array*) – A full array containing binned data to be dynamically binned. Data in this array will be altered by this function.
- **binbounds1** (*numpy array*) – A N x 2 integer array indicating the start and end position of each of N bins in 'binned' array along the first axis.

- **binbounds2** (*numpy array*) – A N x 2 integer array indicating the start and end position of each of N bins in 'binned' array along the second axis.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns None

hifive.fivec_binning.**unbinned_cis_signal**(*fivec*, *region*, *start=None*, *stop=None*, *startfrag=None*, *stopfrag=None*, *datatype='enrichment'*, *arraytype='compact'*, *skipfiltered=False*, *returnmapping=False*)

Create an array of format 'arraytype' and fill with data requested in 'datatype'.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **region** (*int.*) – The index of the region to pull data from.
- **start** (*int.*) – The smallest coordinate to include in the array, measured from fragment midpoints. If both 'start' and 'startfrag' are given, 'start' will override 'startfrag'. If unspecified, this will be set to the midpoint of the first fragment for 'region'. Optional.
- **stop** (*int.*) – The largest coordinate to include in the array, measured from fragment midpoints. If both 'stop' and 'stopfrag' are given, 'stop' will override 'stopfrag'. If unspecified, this will be set to the midpoint of the last fragment plus one for 'region'. Optional.
- **startfrag** (*int.*) – The first fragment to include in the array. If unspecified and 'start' is not given, this is set to the first fragment in 'region'. In cases where 'start' is specified and conflicts with 'startfrag', 'start' is given preference. Optional.
- **stopfrag** (*int.*) – The first fragment not to include in the array. If unspecified and 'stop' is not given, this is set to the last fragment in 'region' plus one. In cases where 'stop' is specified and conflicts with 'stopfrag', 'stop' is given preference. Optional.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are aways in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fragments return value of one. Expected values are returned for 'distance', 'fragment', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance mean values. 'enrichment' also scales both observed and expected by the standard deviation, giving a completely normalized set of values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact', 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N and M are the number of forward and reverse probe fragments, respectively. 'full' returns a square, symmetric array of size N x N x 2 where N is the total number of fragments. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2, where N is the total number of fragments.

- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fragments are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and either a 1d array containing fragment numbers included in the data array if the array is not compact or two 1d arrays containin fragment numbers for forward and reverse fragments if the array is compact is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing data requested with 'datatype'.

hifive.fivec_binning.**unbinned_trans_signal**(*fivec*, *region1*, *region2*, *start1=None*, *stop1=None*, *startfrag1=None*, *stopfrag1=None*, *start2=None*, *stop2=None*, *startfrag2=None*, *stopfrag2=None*, *datatype='enrichment'*, *arraytype='full'*, *skipfiltered=False*, *returnmapping=False*)

Create an array of format 'arraytype' and fill 'binsize' bins with data requested in 'datatype'.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **region1** (*int.*) – The index of the first region to pull data from.
- **region2** (*int.*) – The index of the second region to pull data from.
- **start1** (*int.*) – The coordinate at the beginning of the smallest bin from 'region1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfrag1' mid. If there is a conflict between 'start1' and 'startfrag1', 'start1' is given preference. Optional.
- **stop1** (*int.*) – The largest coordinate to include in the array from 'region1', measured from fragment midpoints. If both 'stop1' and 'stopfrag1' are given, 'stop1' will override 'stopfrag1'. 'stop1' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag1** (*int.*) – The first fragment from 'region1' to include in the array. If unspecified and 'start1' is not given, this is set to the first valid fend in 'region1'. In cases where 'start1' is specified and conflicts with 'startfrag1', 'start1' is given preference. Optional.
- **stopfrag1** (*int.*) – The first fragment not to include in the array from 'region1'. If unspecified and 'stop1' is not given, this is set to the last valid fragment in 'region1' + 1. In cases where 'stop1' is specified and conflicts with 'stopfrag1', 'stop1' is given preference. Optional.
- **start1** – The coordinate at the beginning of the smallest bin from 'region1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfrag1' mid. If there is a conflict between 'start1' and 'startfrag1', 'start1' is given preference. Optional.
- **stop2** (*int.*) – The largest coordinate to include in the array from 'region2', measured from fragment midpoints. If both 'stop2' and 'stopfrag2' are given, 'stop2' will override 'stopfrag2'. 'stop2' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfrag2** (*int.*) – The first fragment from 'region2' to include in the array. If unspecified and 'start2' is not given, this is set to the first valid fend in 'region2'. In cases where 'start2' is specified and conflicts with 'startfrag2', 'start2' is given preference. Optional.
- **stopfrag2** (*int.*) – The first fragment not to include in the array from 'region2'. If unspecified and 'stop2' is not given, this is set to the last valid fragment in 'region2' + 2. In cases where 'stop2' is specified and conflicts with 'stopfrag2', 'stop2' is given preference. Optional.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fragment', 'enrichment', and 'expected'. Observed values are aways in

the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, non-filtered bins return value of 1. Expected values are returned for 'distance', 'fragment', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fragment' uses only fragment correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.

- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact' and 'full'. 'compact' means data are arranged in a N x M x 2 array where N and M are the number of forward and reverse probe fragments, respectively. Two arrays will be returned for this format, the first with forward probe fragments from region1 and reverse probe fragments from region2. The second is the compliment of the first. 'full' returns a square, symmetric array of size N x N x 2 where N is the total number of fragments.
- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fragments are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and mapping information is returned. Otherwise only a data array(s) is returned.

Returns Array in format requested with 'arraytype' containing inter-region data requested with 'datatype'. If 'returnmapping' is True, a list is returned with mapping information. If 'arraytype' is 'full', a single data array and a 1d array of fragments corresponding to rows and columns is returned. If 'arraytype' is 'compact', two data arrays are returned (forward1 by reverse2 and forward2 by reverse1) along with forward and reverse fragment positions for each array for a total of 5 arrays.

hifive.fivec_binning.**write_heatmap_dict**(*fivec*, *filename*, *binsize*, *includetrans=True*, *remove_distance=False*, *arraytype='full'*, *regions=[]*)

Create an h5dict file containing binned interaction arrays, bin positions, and an index of included regions.

Parameters

- **fivec** (FiveC) – A FiveC class object containing fragment and count data.
- **filename** (*str.*) – Location to write h5dict object to.
- **binsize** (*int.*) – Size of bins for interaction arrays. If "binsize" is zero, fragment interactions are returned without binning.
- **includetrans** (*bool.*) – Indicates whether trans interaction arrays should be calculated and saved.
- **remove_distance** (*bool.*) – If 'True', the expected value is calculated including the expected distance mean. Otherwise, only fragment corrections are used.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact' and 'full'. 'compact' means data are arranged in a N x M x 2 array where N is the number of bins, M is the maximum number of steps between included bin pairs, and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2.
- **regions** (*list.*) – If given, indicates which regions should be included. If left empty, all regions are included.

Returns None

CHAPTER 13

The hic_binning module

This is a module contains scripts for generating compact, upper-triangle and full matrices of HiC interaction data.

13.1 Concepts

These functions rely on the `HiC` class in conjunction with the `Fend` and `HiCData` classes.

Data can either be arranged in compact, complete, or flattened (row-major) upper-triangle arrays. Compact arrays are N x M, where N is the number of fends or bins, and M is the maximum distance between fends or bins. This is useful for working with sets of short interactions. Data can be raw, fend-corrected, distance-dependence removed, or enrichment values. Arrays are 3-dimensional with observed values in the first layer of d3, expected values in the second layer of d3. The exception to this is upper-triangle arrays, which are 2d, divinding observed and expected along the second axis.

13.2 API Documentation

hifive.hic_binning.**bin_cis_array**(*hic*, *unbinned*, *fends*, *start=None*, *stop=None*, *binsize=10000*, *binbounds=None*, *arraytype='full'*, *returnmapping=False*)

Create an array of format 'arraytype' and fill 'binsize' bins or bins defined by 'binbounds' with data provided in the array passed by 'unbinned'.

Parameters

- **hic** (`HiC`) – A `HiC` class object containing fend and count data.
- **unbinned** (*numpy array*) – A 2d or 3d array containing data to be binned. Array format will be determined from the number of dimensions.
- **fends** (*numpy array*) – A 1d integer array indicating which position corresponds to which fend in the 'unbinned' array.
- **start** (*int.*) – The coordinate at the beginning of the smallest bin. If unspecified, 'start' will be the first multiple of 'binsize' below the first mid from 'fends'. If 'binbounds' is given, 'start' is ignored. Optional.
- **stop** (*int.*) – The coordinate at the end of the last bin. If unspecified, 'stop' will be the first multiple of 'binsize' above the last mid from 'fends'. If needed, 'stop' is adjusted upward to create a complete last bin. If 'binbounds' is given, 'stop' is ignored. Optional.
- **binsize** (*int.*) – This is the coordinate width of each bin. If binbounds is not None, this value is ignored.

- **binbounds** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins. Any fend not falling in a bin is ignored.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact', 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N is the number of bins, M is the maximum number of steps between included bin pairs, and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2.
- **maxdistance** (*str.*) – This specifies the maximum coordinate distance between bins that will be included in the array. If set to zero, all distances are included.
- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fends are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 2d array of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing binned data requested with 'datatype' pulled from 'unbinned'.

hifive.hic_binning.**bin_cis_signal**(*hic*, *chrom*, *start=None*, *stop=None*, *startfend=None*, *stopfend=None*, *binsize=10000*, *binbounds=None*, *datatype='enrichment'*, *arraytype='full'*, *maxdistance=0*, *returnmapping=False*)

Create an array of format 'arraytype' and fill with data requested in 'datatype' aggregated in bins.

Parameters

- **hic** (HiC) – A HiC class object containing fend and count data.
- **chrom** (*str.*) – The name of a chromosome contained in 'hic'.
- **start** (*int.*) – The coordinate at the beginning of the smallest bin. If unspecified, 'start' will be the first multiple of 'binsize' below the 'startfend' mid. If there is a conflict between 'start' and 'startfend', 'start' is given preference. Optional.
- **stop** (*int.*) – The largest coordinate to include in the array, measured from fend midpoints. If both 'stop' and 'stopfend' are given, 'stop' will override 'stopfend'. If unspecified, this will be set to the midpoint of the last fend plus one for 'chrom'. Optional.
- **startfend** (*int.*) – The first fend to include in the array. If unspecified and 'start' is not given, this is set to the first fend in 'chrom'. In cases where 'start' is specified and conflicts with 'startfend', 'start' is given preference. Optional
- **stopfend** (*str.*) – The first fend not to include in the array. If unspecified and 'stop' is not given, this is set to the last fend in 'chrom' plus one. In cases where 'stop' is specified and conflicts with 'stopfend', 'stop' is given preference. Optional.

- **binsize** (*int.*) – This is the coordinate width of each bin. If binbounds is not None, this value is ignored.
- **binbounds** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins. Any fend not falling in a bin is ignored.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact', 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N is the number of bins, M is the maximum number of steps between included bin pairs, and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2.
- **maxdistance** (*str.*) – This specifies the maximum coordinate distance between bins that will be included in the array. If set to zero, all distances are included.
- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fends are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 2d array of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing binned data requested with 'datatype'.

hifive.hic_binning.**bin_trans_signal**(*hic*, *chrom1*, *chrom2*, *start1=None*, *stop1=None*, *startfend1=None*, *stopfend1=None*, *binbounds1=None*, *start2=None*, *stop2=None*, *startfend2=None*, *stopfend2=None*, *binbounds2=None*, *binsize=1000000*, *datatype='enrichment'*, *returnmapping=False*)

Create a rectangular array and fill with trans data requested in 'datatype' aggregated in bins.

Parameters

- **hic** (HiC) – A HiC class object containing fend and count data.
- **chrom1** (*str.*) – The name of the first chromosome to use.
- **chrom2** (*str.*) – The name of the second chromosome to use.
- **start1** (*int.*) – The coordinate at the beginning of the smallest bin from 'chrom1'. If unspecified, 'start1' will be the first multiple of 'binsize' below the 'startfend1' mid. If there is a conflict between 'start1' and 'startfend1', 'start1' is given preference. Optional.
- **stop1** (*int.*) – The largest coordinate to include in the array from 'chrom1', measured from fend midpoints. If both 'stop1' and 'stopfend1' are given, 'stop1' will override 'stopfend1'. 'stop1' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfend1** (*int.*) – The first fend from 'chrom1' to include in the array. If unspecified and 'start1' is not given, this is set to the first valid fend in 'chrom1'. In cases where 'start1' is specified and conflicts with 'startfend1', 'start1' is given preference. Optional

- **stopfend1** – The first fend not to include in the array from 'chrom1'. If unspecified and 'stop1' is not given, this is set to the last valid fend in 'chrom1' + 1. In cases where 'stop1' is specified and conflicts with 'stopfend1', 'stop1' is given preference. Optional.
- **binbounds1** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins to use for partitioning 'chrom1'. Any fend not falling in a bin is ignored.
- **start2** (*int.*) – The coordinate at the beginning of the smallest bin from 'chrom2'. If unspecified, 'start2' will be the first multiple of 'binsize' below the 'startfend2' mid. If there is a conflict between 'start2' and 'startfend2', 'start2' is given preference. Optional.
- **stop2** (*int.*) – The largest coordinate to include in the array from 'chrom2', measured from fend midpoints. If both 'stop2' and 'stopfend2' are given, 'stop2' will override 'stopfend2'. 'stop2' will be shifted higher as needed to make the last bin of size 'binsize'. Optional.
- **startfend2** (*int.*) – The first fend from 'chrom2' to include in the array. If unspecified and 'start2' is not given, this is set to the first valid fend in 'chrom2'. In cases where 'start2' is specified and conflicts with 'startfend2', 'start2' is given preference. Optional
- **stopfend2** (*str.*) – The first fend not to include in the array from 'chrom2'. If unspecified and 'stop2' is not given, this is set to the last valid fend in 'chrom2' + 1. In cases where 'stop2' is specified and conflicts with 'stopfend2', 'stop1' is given preference. Optional.
- **binbounds2** (*numpy array*) – An array containing start and stop coordinates for a set of user-defined bins to use for partitioning 'chrom2'. Any fend not falling in a bin is ignored.
- **binsize** (*int.*) – This is the coordinate width of each bin. If binbounds is not None, this value is ignored.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and two 2d arrays of N x 4 containing the first fend and last fend plus one included in each bin and first and last coordinates for the first and second chromosomes is returned. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing trans data requested with 'datatype'.

hifive.hic_binning.**dynamically_bin_cis_array**(*unbinned*, *unbinnedpositions*, *binned*, *binbounds*, *minobservations=10*, *searchdistance=0*, *removefailed=True*)

Expand bins in 'binned' to include additional data provided in 'unbinned' as necessary to meet 'minobservations', or 'searchdistance' criteria.

Parameters

- **unbinned** (*numpy array*) – A 2d or 3d array containing data in either compact or upper format to be used for filling expanding bins. Array format will be determined from the number of dimensions.
- **unbinnedpositions** (*numpy array*) – An 1d integer array indicating the mid-point of each bin in 'unbinned' array.

- **binned** (*numpy array*) – A 2d or 3d array containing binned data in either compact or upper format to be dynamically binned. Array format will be determined from the number of dimensions. Data in this array will be altered by this function.
- **binbounds** (*numpy array*) – An integer array indicating the start and end position of each bin in 'binned' array. This array should be N x 2, where N is the number of intervals in 'binned'.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns None

hifive.hic_binning.**dynamically_bin_trans_array**(*unbinned*, *unbinnedpositions1*, *unbinnedpositions2*, *binned*, *binbounds1*, *binbounds2*, *minobservations=10*, *searchdistance=0*, *removefailed=False*)

Expand bins in 'binned' to include additional data provided in 'unbinned' as necessary to meet 'minobservations', or 'searchdistance' criteria.

Parameters

- **unbinned** (*numpy array*) – A 3d array containing data to be used for filling expanding bins. This array should be N x M x 2, where N is the number of bins or fends from the first chromosome and M is the number of bins or fends from the second chromosome.
- **unbinnedpositions1** (*numpy array*) – An 1d integer array indicating the mid-point of each bin or fend in the 'unbinned' array along the first axis.
- **unbinnedpositions2** (*numpy array*) – An 1d integer array indicating the mid-point of each bin or fend in the 'unbinned' array along the second axis.
- **binned** (*numpy array*) – A 3d array containing binned data to be dynamically binned. This array should be N x M x 2, where N is the number of bins from the first chromosome and M is the number of bins from the second chromosome. Data in this array will be altered by this function.
- **binbounds1** (*numpy array*) – An integer array indicating the start and end position of each bin from the first chromosome in the 'binned' array. This array should be N x 2, where N is the size of the first dimension of 'binned'.
- **binbounds2** (*numpy array*) – An integer array indicating the start and end position of each bin from the second chromosome in the 'binned' array. This array should be N x 2, where N is the size of the second dimension of 'binned'.
- **minobservations** (*int.*) – The fewest number of observed reads needed for a bin to counted as valid and stop expanding.
- **searchdistance** (*int.*) – The furthest distance from the bin minpoint to expand bounds. If this is set to zero, there is no limit on expansion distance.
- **removefailed** (*bool.*) – If a non-zero 'searchdistance' is given, it is possible for a bin not to meet the 'minobservations' criteria before stopping looking. If this occurs and 'removefailed' is True, the observed and expected values for that bin are zero.

Returns None

hifive.hic_binning.**unbinned_cis_signal**(*hic*, *chrom*, *start=None*, *stop=None*, *startfend=None*, *stopfend=None*, *datatype='enrichment'*, *arraytype='compact'*, *maxdistance=0*, *skipfiltered=False*, *returnmapping=False*)

Create an array of format 'arraytype' and fill with data requested in 'datatype'.

Parameters

- **hic** (HiC) – A HiC class object containing fend and count data.
- **chrom** (*str.*) – The name of a chromosome contained in 'hic'.
- **start** (*int.*) – The smallest coordinate to include in the array, measured from fend midpoints. If both 'start' and 'startfend' are given, 'start' will override 'startfend'. If unspecified, this will be set to the midpoint of the first fend for 'chrom'. Optional.
- **stop** (*int.*) – The largest coordinate to include in the array, measured from fend midpoints. If both 'stop' and 'stopfend' are given, 'stop' will override 'stopfend'. If unspecified, this will be set to the midpoint of the last fend plus one for 'chrom'. Optional.
- **startfend** (*int.*) – The first fend to include in the array. If unspecified and 'start' is not given, this is set to the first fend in 'chrom'. In cases where 'start' is specified and conflicts with 'startfend', 'start' is given preference. Optional
- **stopfend** (*str.*) – The first fend not to include in the array. If unspecified and 'stop' is not given, this is set to the last fend in 'chrom' plus one. In cases where 'stop' is specified and conflicts with 'stopfend', 'stop' is given preference. Optional.
- **datatype** (*str.*) – This specifies the type of data that is processed and returned. Options are 'raw', 'distance', 'fend', 'enrichment', and 'expected'. Observed values are always in the first index along the last axis, except when 'datatype' is 'expected'. In this case, filter values replace counts. Conversely, if 'raw' is specified, unfiltered fends return value of one. Expected values are returned for 'distance', 'fend', 'enrichment', and 'expected' values of 'datatype'. 'distance' uses only the expected signal given distance for calculating the expected values, 'fend' uses only fend correction values, and both 'enrichment' and 'expected' use both correction and distance mean values.
- **arraytype** (*str.*) – This determines what shape of array data are returned in. Acceptable values are 'compact', 'full', and 'upper'. 'compact' means data are arranged in a N x M x 2 array where N is the number of bins, M is the maximum number of steps between included bin pairs, and data are stored such that bin n,m contains the interaction values between n and n + m + 1. 'full' returns a square, symmetric array of size N x N x 2. 'upper' returns only the flattened upper triangle of a full array, excluding the diagonal of size (N * (N - 1) / 2) x 2.
- **maxdistance** (*str.*) – This specifies the maximum coordinate distance between bins that will be included in the array. If set to zero, all distances are included.
- **skipfiltered** (*bool.*) – If 'True', all interaction bins for filtered out fends are removed and a reduced-size array is returned.
- **returnmapping** (*bool.*) – If 'True', a list containing the data array and a 1d array containing fend numbers included in the data array is return. Otherwise only the data array is returned.

Returns Array in format requested with 'arraytype' containing data requested with 'datatype'.

hifive.hic_binning.**write_heatmap_dict**(*hic*, *filename*, *binsize*, *includetrans=True*, *removedistance=False*, *chroms=[]*)

Create an h5dict file containing binned interaction arrays, bin positions, and an index of included chromosomes. This function is MPI compatible.

Parameters

- **hic** (HiC) – A HiC class object containing fend and count data.
- **filename** (*str.*) – Location to write h5dict object to.
- **binsize** (*int.*) – Size of bins for interaction arrays.
- **includetrans** (*bool.*) – Indicates whether trans interaction arrays should be calculated and saved.
- **remove_distance** (*bool.*) – If 'True', the expected value is calculated including the expected distance mean. Otherwise, only fend corrections are used.
- **chroms** (*list*) – A list of chromosome names indicating which chromosomes should be included. If left empty, all chromosomes are included. Optional.

Returns None

CHAPTER 14

The plotting module

This is a module contains scripts for generating plots from compact and full matrices of interaction data.

14.1 Concepts

These functions take either compact, upper-triangle, or full 3d data matrices.

Data can either be arranged in compact, upper-triangle, or complete (rectangular) arrays. With HiC data, compact arrays are N x M x 2, where N is the number of fends or bins, and M is the maximum distance between fends or bins. This is useful for working with sets of short interactions. When using 5C data, the compact format is an N x M x 2 array where N is the number of forward primers and M is the number of reverse primers. Data can be raw, fend-corrected, distance-dependence removed, or enrichment values. Arrays are 3-dimensional with observed values in the first layer of d3, expected values in the second layer of d3.

14.2 API Documentation

hifive.plotting.**plot_compact_array**(*data*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap from a HiC compact array.

Parameters

- **data** (*numpy array*) – A three-dimensional compact array of HiC interaction data.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers

range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.

- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns PIL bitmap object.

hifive.plotting.**plot_diagonal_from_compact_array**(*data*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap from a HiC compact array, plotting only the upper triangle rotated 45 degrees counterclockwise.

Parameters

- **data** (*numpy array*) – A three-dimensional compact array of HiC interaction data.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns PIL bitmap object.

hifive.plotting.**plot_fivec_compact_heatmap_dict**(*filename*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *regions=[]*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap in a compact from a 5C heatmap h5dict file.

This plots the data in a 5C compact format such that the rows correspond to positive-strand primers and columns correspond to negative-strand primers.

Parameters

- **filename** – Location of a heatmap h5dict containing 5C data arrays.

- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **regions** (*list*) – If specified, only the indicated regions are plotted. Otherwise all regions present in the h5dict are plotted.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns `PIL` bitmap object.

`hifive.plotting.`**`plot_fivec_full_heatmap_dict`**(*filename*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *regions=[]*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap in a full format from a 5C heatmap h5dict file.

This plots the data in a full format such that the rows and columns contain both orientations of primers.

Parameters

- **filename** – Location of a heatmap h5dict containing 5C data arrays.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **regions** (*list*) – If specified, only the indicated regions are plotted. Otherwise all regions present in the h5dict are plotted.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.

- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns PIL bitmap object.

hifive.plotting.**plot_full_array**(*data*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap from a 5C or HiC full array.

Parameters

- **data** (*numpy array*) – A three-dimensional compact array of interaction data.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns PIL bitmap object.

hifive.plotting.**plot_hic_heatmap_dict**(*filename*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *chroms=[]*, *min_color=(0.0, 0.0, 1.0)*, *mid_color=(1.0, 1.0, 1.0)*, *max_color=(1.0, 0.0, 0.0)*)

Fill in and rescale bitmap from a HiC heatmap h5dict file.

Parameters

- **filename** (*str.*) – File name of heatmap h5dict containing binned data arrays.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **chroms** (*list*) – A list of chromosome names to include in the plot. If left empty, all chromosomes present in the heatmap file will be plotted.

- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns `PIL` bitmap object.

`hifive.plotting.`**`plot_key`**(*min_score*, *max_score*, *height*, *width*, *labelformat='%0.2f'*, *orientation='left'*, *num_ticks=5*, *min_color=(0.0*, *0.0*, *1.0)*, *mid_color=(1.0*, *1.0*, *1.0)*, *max_color=(1.0*, *0.0*, *0.0)*, *labelattr=None*, *log_display=True*)

Create a key including color gradient and labels indicating associated values, returning a `pyx` canvas.

Parameters

- **min_score** (*float*) – The minimum value of the key scale.
- **max_score** (*float*) – The maximum value of the key scale.
- **height** (*float*) – The height of the gradient bar in whatever units `pyx` is using.
- **width** (*float*) – The width of the gradient bar in whatever units `pyx` is using.
- **labelformat** (*str.*) – A string denoting the format for displaying number labels using the string formatting style from Python <= 2.6.
- **orientation** – Indicates where labels are placed relative to gradient bar. This parameter will accept 'left', 'right', 'top', and 'bottom'.
- **num_ticks** (*int.*) – Indicates how many evenly-spaced tick marks and associated labels to insert. This can be zero for no labels or greater than one. Labels are inserted at the minimum and maximum values first with remaining ticks occuring evenly distributed between the extremes.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.
- **labelattr** (*str.*) – A list of pyx attributes to be passed to the text function.
- **log_display** (*bool.*) – If True, min_score and max_score are taken to be logged values and so labels are evenly spaced in log space but converted to normal space for display.

Returns `Pxy` canvas object.

`hifive.plotting.`**`plot_upper_array`**(*data*, *maxscore=None*, *minscore=None*, *symmetricscaling=True*, *logged=True*, *min_color=(0.0*, *0.0*, *1.0)*, *mid_color=(1.0*, *1.0*, *1.0)*, *max_color=(1.0*, *0.0*, *0.0)*)

Fill in and rescale bitmap from a 5C or HiC upper array.

Parameters

- **data** (*numpy array*) – A two-dimensional compact array of interaction data.
- **maxscore** (*float*) – A ceiling value to cutoff scores at for plot color.
- **minscore** (*float*) – A floor value to cutoff scores at for plot color.
- **symmetricscaling** (*bool.*) – Indicates whether to recenter data for scaling or maintain scores about zero.
- **logged** (*bool.*) – Indicates whether to use log values of scores for color values.
- **min_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the minimum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with max_color and optionally mid_color.
- **mid_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the middle plot value, respectively. Numbers range from 0.0 to 1.0. This can be set to None to create a gradient ranging from min_color to max_color or to a tuple to create a divergent gradient.
- **max_color** (*tuple*) – This is a tuple containing three numbers representing the red, green, and blue component of the color associated with the maximum plot value, respectively. Numbers range from 0.0 to 1.0. This variable is used to create a color gradient for plotting along with min_color and optionally mid_color.

Returns `PIL` bitmap object.

CHAPTER 15

Additional Scripts

To make using HiFive easier, several scripts are included to perform all of the standard functions.

15.1 create_fragmentset.py

This script produces a Fragment h5dict from a BED file containing restriction fragment boundaries and primer names targeting them. This script can be called as follows:

```
> python create_fragmentset.py BED_FILE OUT_FILE [GENOME RE]
```

Arguments:

- **BED_FILE** (*str.*) - File containing 5C targeted fragments and primer names in BED format.
- **OUT_FILE** (*str.*) - File name to write Fragment h5dict to.
- **GENOME** (*str.*) - Name of the genome.
- **RE** (*str.*) - Name of the restriction enzyme.

Returns: None

15.2 create_fendset.py

This script produces a Fend h5dict from a HiCPipe-compatible fend file or BED file containing restriction fragment boundaries or sites. This script can be called as follows:

```
> python create_fendset.py FEND_FILE OUT_FILE [GENOME RE]
```

Arguments:

- **FEND_FILE** (*str.*) - File containing restriction fragment data in HiCPipe-compatible or BED format.
- **OUT_FILE** (*str.*) - File name to write Fend h5dict to.
- **GENOME** (*str.*) - Name of the genome.
- **RE** (*str.*) - Name of the restriction enzyme.

Returns: None

15.3 create_fivec_dataset.py

This script produces a `FiveCData` h5dict from a set of counts files or BAM files containing mapped paired-end reads. This script can be called as follows:

```
> python create_fivec_dataset.py FRAG_FILE DATA_FILE_1[,...,DATA_FILE_N] OUT_FILE
```

Arguments:

- **FRAG_FILE** (*str.*) - File name of `Fragment` h5dict to link with data.
- **DATA_FILE_1...** (*str.*) - A comma-separated list of either count file names or BAM file prefices.
- **OUT_FILE** (*str.*) - File name to write `FiveCData` h5dict to.

Returns: None

15.4 create_hic_dataset.py

This script produces a `HiCData` h5dict from a set of BAM files containing mapped paired-end reads, HiCPipe-compatible MAT-formatted data files or raw paired-coordinate text files. This script can be called as follows:

```
> python create_hic_dataset.py FEND_FILE DATA_FILE_1[,...,DATA_FILE_N] OUT_FILE MAX_INSERT
```

Arguments:

- **FEND_FILE** (*str.*) - File name of `Fend` h5dict to link with data.
- **DATA_FILE_1[,...,DATA_FILE_N]** (*str.*) - A comma-separated list of either BAM file prefices, raw coordinate read pairs or HiCPipe-compatible MAT files.
- **OUT_FILE** (*str.*) - File name to write `HiCData` h5dict to.
- **MAX_INSERT** (*int.*) - Integer specifying the maximum distance sum from each mapped end to restriction site.

Returns: None

15.5 combine_replicates.py

This script combines reads from multiple replicate `HiCData` h5dicts and creates a new h5dict. This script can be called as follows:

```
> python combine_replicates.py REP_FILE_1,REP_FILE_2[,...,REP_FILE_N] OUT_FILE
```

Arguments:

- **REP_FILE_1,REP_FILE2** (*str.*) - A comma-separated list of `HiCData` h5dict files.
- **OUT_FILE** (*str.*) - File name to write `HiCData` h5dict to.

Returns: None

15.6 data2mat.py

This script exports read data from a `HiCData` h5dict into a HiCPipe-compatible MAT-formatted text file. This script can be called as follows:

```
> python data2mat.py DATA_FILE OUT_FILE
```

Arguments:

- **DATA_FILE** (*str.*) - File name of `HiCData` h5dict.
- **OUT_FILE** (*str.*) - File name to write HiCPipe-compatible MAT-formatted data to.

Returns: None

15.7 create_fivec_set.py

This script creates a `FiveC` h5dict analysis object, filters fragments, and calculates the distance dependence function. This script can be called as follows:

```
> python create_fivec_set.py DATA_FILE OUT_FILE MIN_INTERACTIONS
```

Arguments:

- **DATA_FILE** (*str.*) - File name of `FiveCData` h5dict to link with analysis.
- **OUT_FILE** (*str.*) - File name to write `FiveC` h5dict to.
- **MIN_INTERACTIONS** (*int.*) - Minimum number of interactions needed for valid fragment.

Returns: None

15.8 create_hic_set.py

This script creates a `HiC` h5dict analysis object, filters fends, and calculates the distance dependence function. This script can be called as follows:

```
> python create_hic_set.py DATA_FILE HIC_FILE MIN_INTERACTIONS MAX_DIST MIN_SIZE NUM_BINS SMOOTHED
```

Arguments:

- **DATA_FILE** (*str.*) - File name of `HiCData` h5dict to link with analysis.
- **OUT_FILE** (*str.*) - File name to write `HiC` h5dict to.
- **MIN_INTERACTIONS** (*int.*) - Minimum number of interactions needed for valid fend.
- **MAX_DIST** (*int.*) - The largest interaction distance to be included for filtering fends.
- **MIN_SIZE** (*int.*) - The smallest interaction distance bin size for distance function.
- **NUM_BINS** (*int.*) - The number of bins to partion interaction distance range into for distance function.
- **SMOOTHED** (*int.*) - Number of adjacent bins to include for smoothing of distance function line.

Returns: None

Note: This function is MPI compatible.

15.9 learn_fivec_normalization.py

This script learns fragment correction values for a `FiveC` analysis object. This script can be called as follows:

```
> python learn_fivec_normalization.py FIVEC_FILE RATE BURNIN ANNEALING MAX_DIST RECALC DISPLAY
```

Arguments:

- **FIVEC_FILE** (*str.*) - File name of `FiveC` h5dict to analyze.
- **RATE** (*float*) - Percent of gradient to use for updating parameter values.
- **BURNIN** (*int.*) - Number of iterations to run burn-in phase for.
- **ANNEALING** (*int.*) - Number of iterations to run annealing phase for.
- **MAX_DIST** (*int.*) - Maximum interaction distance to include in learning.
- **RECALC** (*int.*) - Number of iterations to wait between recalculating distance function parameters.
- **DISPLAY** (*int.*) - Number of iterations to wait before explicitly calculating cost and updating display.

Returns: None

15.10 learn_fivec_normalization_express.py

This script learns fragment correction values for a `FiveC` analysis object using the approximation approach. This script can be called as follows:

```
> python learn_fivec_normalization_express.py FIVEC_FILE ITERATIONS REMOVE_DIST RECALC
```

Arguments:

- **FIVEC_FILE** (*str.*) - File name of `FiveC` h5dict to analyze.
- **ITERATIONS** (*int.*) - Number of iterations to run learning for.
- **REMOVE_DIST** (*bool.*) - Specifies whether to remove distance-dependent portion of the signal prior to learning.
- **RECALC** (*int.*) - Number of iterations to wait between recalculating distance function parameters.

Returns: None

15.11 learn_hic_normalization.py

This script learns fend correction values for a `HiC` analysis object. This script can be called as follows:

```
> python learn_hic_normalization.py HIC_FILE BURNIN ANNEALING MAX_DIST RECALC RATE DISPLAY
```

Arguments:

- **HIC_FILE** (*str.*) - File name of HiC h5dict to analyze.
- **BURNIN** (*int.*) - Number of iterations to run burn-in phase for.
- **ANNEALING** (*int.*) - Number of iterations to run annealing phase for.
- **MAX_DIST** (*int.*) - Maximum interaction distance to include in learning.
- **RECALC** (*int.*) - Number of iterations to wait between recalculating distance function parameters.
- **RATE** (*float*) - Percent of gradient to use for updating parameter values.
- **DISPLAY** (*int.*) - Number of iterations to wait before explicitly calculating cost and updating display.

Returns: None

Note: This function is MPI compatible.

15.12 learn_hic_normalization_express.py

This script learns fend correction values for a HiC analysis objectusing the approximation approach. This script can be called as follows:

```
> python learn_hic_normalization_express.py HIC_FILE ITERATIONS MIN_INT MIN_DIST USE_READS REMOVE_DIS
```

Arguments:

- **HIC_FILE** (*str.*) - File name of HiC h5dict to analyze.
- **ITERATIONS** (*int.*) - Number of iterations to run learning for.
- **MIN_INT** (*int.*) - Minimum number of interactions for fend filtering, if refiltering is required.
- **MIN_DIST** (*int.*) - Minimum interaction distance to include for learning.
- **USE_READS** (*str.*) - Which set of reads, 'cis', 'trans', or 'both', to use for learning.
- **REMOVE_DISTANCE** (*bool.*) - Specifies whether to remove distance-dependent portion of the signal prior to learning.
- **RECALC** (*int.*) - Number of iterations to wait between recalculating distance function parameters.

Returns: None

Note: This function is MPI compatible.

15.13 create_hic_heatmap_h5dict.py

This script creates an h5dict file containing binned heatmaps from a HiC h5dict. This script can be called as follows:

```
> python create_hic_heatmap_h5dict.py HIC_FILE OUT_FILE BINSIZE INCLUDE_TRANS REMOVE_DISTANCE CHROMS
```

- **HIC_FILE** (*str.*) - File name of a HiC h5dict to pull data from.
- **OUT_FILE** (*str.*) - File name of heatmap h5dict to write data to.
- **BINSIZE** (*int.*) - Size of bins, in base pairs, to group data into.
- **INCLUDE_TRANS** (*bool.*) - Specifies whether to find inter-chromosome interactions.
- **REMOVE_DISTANCE** (*bool.*) - Specifies whether to remove distance-dependent portion of signal.
- **CHROMS** (*str.*) - Comma-separated list of chromosomes to find heatmaps for.

Returns: None

15.14 find_hic_BI.py

This scripts takes a HiC file and calculates a set of BI scores. This script can be called as follows:

```
> python find_hic_BI.py HIC_FILE OUT_FILE WIDTH HEIGHT WINDOW MINCOUNT SMOOTHING [CHROM_1,...,CHROM_N
```

Arguments:

- **HIC_FILE** (*str.*) - H5dict created by the HiC class.
- **OUT_FILE** (*str.*) - File name for the new BI h5dict created by this script.
- **WIDTH** (*int.*) - Integer specifying the width about each boundary point.
- **HEIGHT** (*int.*) - Integer specifying the height of bins extending across each window.
- **WINDOW** (*int.*) - Integer specifying the window around each boundary point.
- **MINCOUNT** (*int.*) - Minimum number of valid bin pairs needed to find BI value.
- **SMOOTHING** (*int.*) - Integer specifying the width of smoothing weights.
- **CHROM_1,...CHROM_N** (*str.*) - A comma-separated list of chromosome names to include in the analysis. Optional.

Returns: None

Note: This function is MPI compatible.

15.15 combine_BIs.py

This script takes two BI files with different coordinates, such as would be created by two different restriction enzymes, annd comnines the data to create a composite set of scores. The script can be called as follows:

```
> python combine_BIs.py BI_FILE_1 BI_FILE_2 OUT_FILE SMOOTHING CHROM_1[,...,CHROM_N]
```

Arguments:

- **BI_FILE_1** (*str.*) - The path of an h5dict file created by a BI object.
- **BI_FILE_2** (*str.*) - The path of an h5dict file created by a BI object.

- **OUT_FILE** (*str.*) - The path to write the new h5dict file to.
- **SMOOTHING** (*int.*) - The width, in base pairs, for smoothing BI scores.
- **CHROM_1[..,CHROM_N]** (*str.*) - A comma-separated list of chromosome names to include in the analysis.

Returns: None

Note: This function is MPI compatible.

15.16 model_single_chr_BI.py

This script bins data from a `HiC` h5dict using peaks calls from a `BI` object to partition signal, dynamically bins the data, and creates a 3D model using a PCA dimensionality reduction. The script can be called as follows:

```
> python model_single_chr_BI.py HIC_FILE BI_FILE OUT_PREFIX CUTOFF MIN_OBS EXP_BINSIZE CHROM
```

Arguments:

- **HIC_FILE** (*str.*) - File name of `HiC` h5dict to pull data from.
- **BI_FILE** (*str.*) - File name of `BI` h5dict to find boundaries for partitioning from.
- **OUT_PREFIX** (*str.*) - File prefix for all output files of script.
- **CUTOFF** (*float*) - Criteria for calling BI peaks.
- **MIN_OBS** (*int.*) - Minimum number of observations for valid dynamic bins.
- **EXP_BINSIZE** (*int.*) - Size of bins for additional data used for dynamic bin expansion. This may be set to zero for unbinned data.
- **CHROM** (*str.*) - Name of chromosome to model.

Returns: None

15.17 model_single_chr_binned.py

This script bins data from a `HiC` h5dict, dynamically bins the data, and creates a 3D model using a PCA dimensionality reduction. The script can be called as follows:

```
> python model_single_chr_BI.py HIC_FILE OUT_PREFIX BINSIZE EXP_BINSIZE CHROM
```

Arguments:

- **HIC_FILE** (*str.*) - File name of `HiC` h5dict to pull data from.
- **OUT_PREFIX** (*str.*) - File prefix for all output files of script.
- **BIN_SIZE** (*str.*) - Size of bins, in base pairs, to group data into.
- **MIN_OBS** (*int.*) - Minimum number of observations for valid dynamic bins.
- **EXP_BINSIZE** (*int.*) - Size of bins for additional data used for dynamic bin expansion. This may be set to zero for unbinned data.
- **CHROM** (*str.*) - Name of chromosome to model.

Returns: None

15.18 model_whole_genome_BI.py

This script bins data from a `HiC` h5dict using peaks calls from a `BI` object to partition signal, dynamically bins the data, and creates a 3D model using a PCA dimensionality reduction. The script can be called as follows:

```
> python model_whole_genome_BI.py HIC_FILE BI_FILE OUT_PREFIX CUTOFF MIN_OBS CIS_SCALING EXP_BINSIZE
```

Arguments:

- **HIC_FILE** (*str.*) - File name of `HiC` h5dict to pull data from.
- **BI_FILE** (*str.*) - File name of `BI` h5dict to find boundaries for partitioning from.
- **OUT_PREFIX** (*str.*) - File prefix for all output files of script.
- **CUTOFF** (*float*) - Criteria for calling BI peaks.
- **MIN_OBS** (*int.*) - Minimum number of observations for valid dynamic bins.
- **CIS_SCALING** (*float*) - Scaling factor to adjust cis interactions by prior to modeling.
- **EXP_BINSIZE** (*int.*) - Size of bins for additional data used for dynamic bin expansion. This may be set to zero for unbinned data.
- **CHROMS** (*str.*) - Comma-separated list of names of chromosomes to model.

Returns: None

Note: This function is MPI compatible.

15.19 model_whole_genome_binned.py

This script bins data from a `HiC` h5dict, dynamically bins the data, and creates a 3D model using a PCA dimensionality reduction. The script can be called as follows:

```
> python model_whole_genome_binned.py HIC_FILE OUT_PREFIX BIN_SIZE MIN_OBS CIS_SCALING EXP_BINSIZE CH
```

Arguments:

- **HIC_FILE** (*str.*) - File name of `HiC` h5dict to pull data from.
- **OUT_PREFIX** (*str.*) - File prefix for all output files of script.
- **BIN_SIZE** (*str.*) - Size of bins, in base pairs, to group data into.
- **MIN_OBS** (*int.*) - Minimum number of observations for valid dynamic bins.
- **CIS_SCALING** (*float*) - Scaling factor to adjust cis interactions by prior to modeling.
- **EXP_BINSIZE** (*int.*) - Size of bins for additional data used for dynamic bin expansion. This may be set to zero for unbinned data.
- **CHROMS** (*str.*) - Comma-separated list of names of chromosomes to model.

Returns: None

Note: This function is MPI compatible.

CHAPTER 16

Indices and tables

- *genindex*
- *modindex*
- *search*

h

B

C

D

E

F

H

L

P

S

T

U

W