BandHiC

Weibing Wang

Jul 15, 2025

USER GUIDE

Welcome to the official documentation for the **BandHiC** package. This guide includes tutorials, installation instructions, and a complete API reference.

CHAPTER

ONE

INSTALLATION

This section describes how to install **BandHiC**.

1.1 Requirements

- Python >= 3.8
- NumPy >= 1.20
- SciPy >= 1.6
- cooler
- straw

You can install these using pip:

```
pip install numpy scipy cooler straw
```

Install from PyPI (if available):

```
pip install bandhic
```

Or install from source:

```
git clone https://github.com/xdwwb/BandHiC.git
cd BandHiC
pip install -e .
```

To verify the installation:

```
import bandHiC
print(bandHiC.__version__)
```

CHAPTER

TWO

USAGE

This section shows basic usage of the **BandHiC** package.

Read a .hic file and construct a band matrix:

```
from bandHiC import read_hic_chr

mat = read_hic_chr("GM12878.hic", method="straw", chrom="chr1", resolution=10000, diag_
↪num=200)
print(mat.shape)
```

Perform a basic matrix operation:

```
mat_clipped = mat.clip_copy(0, 100)
mat_sum = mat.sum(axis="row")
```

Visualize matrix:

```
import matplotlib.pyplot as plt
plt.imshow(mat.todense(), cmap="Reds")
plt.colorbar()
plt.title("BandHiC Matrix")
plt.show()
```

2.1 Load a .cool file

You can also load contact matrices from *.cool* files using the *cooler_chr* function:

```
from bandHiC import cooler_chr
mat = cooler_chr("example.cool", "chr1", resolution=10000, diag_num=150)
print(mat.shape)
```

2.2 Mathematical operations

BandHiC supports NumPy-like reductions such as *mean*, *sum*, *max*, etc., on rows, columns, or diagonals:

```
row_mean = mat.mean(axis="row")
col_max = mat.max(axis="col")
diag_sum = mat.sum(axis="diag")
```

2.3 Masking operations

You can apply or modify the internal mask. To retrieve data while ignoring masked values:

```
values = mat.get_values_ignore_mask(from_id, to_id)
```

2.4 Save and reload

BandHiC matrices can be saved and loaded as compressed *.npz* files:

```
mat.dump("matrix.npz")

from bandHiC import band_hic_matrix
new_mat = band_hic_matrix.load("matrix.npz")
```

2.5 Advanced matrix operations

BandHiC also supports NumPy-like elementwise operations via overloaded operators:

```
mat2 = mat * 2 - 5
mat3 = mat + mat2
```

You can also use NumPy ufuncs:

```
import numpy as np
log_mat = np.log1p(mat)
clipped = np.clip(mat, 0, 100)
```

Iterating over windows or rows:

```
for window in mat.iterwindows(width=10, step=5):
    print(window.shape)

for row in mat.iterrows():
    print(row.mean())
```

2.6 Matrix combination and manipulation

BandHiC matrices can be combined using standard operations:

```
combined = (mat + mat2) / 2
diff = mat - mat2
normalized = (mat - mat.mean(axis="diag")) / mat.std(axis="diag")
```

These operations preserve the band structure and mask where applicable.

2.7 Command-line usage (optional)

If you have scripts that use BandHiC and want to apply them in batch mode:

```
python scripts/compute_band_features.py --input GM12878.hic --chrom chr1 --resolution␣
↪10000 --diag_num 200
```

You can integrate BandHiC into workflows using Snakemake, Makefile, or shell scripts.

CHAPTER

THREE

BANDHIC TUTORIAL (BASIC)

3.1 Overview

BandHiC enables efficient analysis of Hi-C matrices using banded matrix storage and computation.

1. What is a band matrix?
2. Why not use dense / sparse storage?
3. Typical analysis flow:
    - Load data
    - Band conversion
    - Filtering / masking
    - Downstream analysis (e.g., domain detection)

3.2 Example: From cooler file

```
from bandHiC import cooler_chr
mat = cooler_chr("data.cool", "chr1", resolution=5000, diag_num=150)

# Use matrix
mat.mean(axis="col")
```

3.3 Example: Masking and visualizing

BandHiC matrices may contain masked regions, such as unmappable genomic bins. You can visualize the masked structure:

```
import matplotlib.pyplot as plt
masked = mat.todense()
plt.imshow(masked, cmap="Greys", interpolation="none")
plt.title("Masked BandHiC Matrix")
plt.show()
```

3.4 Example: Diagonal operations

Because BandHiC is optimized for diagonal band matrices, you can directly access and compute over diagonals:

```
main_diag = mat.diag(0)
offset_diag = mat.diag(3)
mat.set_diag(0, main_diag * 0.8)

diag_avg = mat.mean(axis="diag")
```

3.5 Saving and reloading

Matrices can be serialized to disk and reloaded:

```
mat.dump("demo_matrix.npz")

from bandHiC import band_hic_matrix
reloaded = band_hic_matrix.load("demo_matrix.npz")
print(reloaded.shape)
```

3.6 Window iteration

You can iterate through sliding windows over the matrix:

```
for window in mat.iterwindows(width=5, step=2):
    print(window.shape, window.mean())
```

3.7 Combining multiple matrices

You can combine two BandHiC matrices from the same chromosome using arithmetic operations:

```
from bandHiC import read_hic_chr

mat1 = read_hic_chr("replicate1.hic", method="straw", chrom="chr1", resolution=10000,␣
↪diag_num=200)
mat2 = read_hic_chr("replicate2.hic", method="straw", chrom="chr1", resolution=10000,␣
↪diag_num=200)

average = (mat1 + mat2) / 2
diff = mat1 - mat2
```

3.8 Normalization example

BandHiC allows simple normalization of diagonal signals:

```
diag_mean = mat.mean(axis="diag")
mat_normalized = (mat - diag_mean) / mat.std(axis="diag")
```

3.9 Batch processing use case

If you process many chromosomes or resolutions, encapsulate logic in functions:

```
def load_and_reduce(file, chrom, resolution):
    mat = read_hic_chr(file, method="straw", chrom=chrom, resolution=resolution, diag_
↪num=200)
    return mat.mean(axis="col")

for chrom in ["chr1", "chr2", "chr3"]:
    avg_col = load_and_reduce("sample.hic", chrom, 10000)
    print(f"{chrom} column average:", avg_col[:5])
```

3.10 Domain detection (basic workflow)

BandHiC can serve as a lightweight backend for detecting domain boundaries using smoothed diagonal profiles.

Step 1: Extract diagonal band statistics

```
profile = mat.mean(axis="diag")
```

Step 2: Smooth the profile (e.g., moving average)

```
import numpy as np
window_size = 5
kernel = np.ones(window_size) / window_size
smoothed = np.convolve(profile, kernel, mode="same")
```

Step 3: Identify domain boundaries (e.g., local minima)

```
from scipy.signal import argrelextrema
minima = argrelextrema(smoothed, np.less)[0]
print("Boundary candidates:", minima[:10])
```

You can further extend this by integrating change point detection or filtering by signal amplitude.

3.11 Visualizing domain signal and boundaries

You can visualize the smoothed profile and its inferred domain boundaries:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.plot(smoothed, label="Smoothed Diagonal Signal", color="steelblue")
plt.scatter(minima, smoothed[minima], color="red", label="Inferred Boundaries")
plt.xlabel("Diagonal Offset")
plt.ylabel("Average Contact")
plt.title("Domain Boundary Detection Profile")
plt.legend()
plt.tight_layout()
plt.show()
```

3.12 Overlaying on heatmap

You can also overlay the detected boundaries on the matrix heatmap:

```
dense = mat.todense()
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(dense, cmap="Reds", interpolation="none")
for d in minima:
    ax.axline((0, d), slope=1, color="blue", linestyle="--", linewidth=1)
plt.title("Hi-C Matrix with Diagonal Boundaries")
plt.show()
```

CHAPTER

FOUR

BANDHIC API REFERENCE

This page provides a full API overview of the BandHiC module, including the core band_hic_matrix class and its utilities.

The *bandhic* module defines data structures and utilities for efficiently storing and manipulating Hi-C contact matrices in a banded form. It enables NumPy-compatible operations, matrix masking, diagonal reduction, file I/O, and domain-level interaction analysis.

Core components: - *band_hic_matrix*: A memory-efficient matrix class supporting banded Hi-C data. - Utility functions: Loaders (*read_hic_chr*, *cooler_chr*) and constructors (*ones*, *zeros*, etc.).

4.1 Class Summary

bandhic.band_hic_matrix(contacts[, ...])	Symmetric banded matrix stored in upper-triangular format. This storage format is motivated by high-resolution Hi-C data characteristics: 1. Symmetry of contact maps. 2. Interaction frequency concentrated near the diagonal; long-range contacts are sparse (mostly zero). 3. Contact frequency decays sharply with genomic distance. By storing only the main and a fixed number of super-diagonals as columns of a band matrix (diagonal-major storage: diagonal k stored in column k), we drastically reduce memory usage while enabling random access to Hi-C contacts. Additionally, mask and mask_row_col arrays track invalid or masked contacts to support downstream analysis.

4.1.1 bandhic.band_hic_matrix

class bandhic.**band_hic_matrix**(*contacts: coo_array | coo_matrix | tuple | ndarray*, *diag_num: int = 1*, *mask_row_col: ndarray | None = None*, *mask: Tuple[ndarray, ndarray] | None = None*, *dtype: type | None = None*, *default_value: int | float = 0*, *band_data_input: bool = False*)

Symmetric banded matrix stored in upper-triangular format. This storage format is motivated by high-resolution Hi-C data characteristics:

1. Symmetry of contact maps.
2. Interaction frequency concentrated near the diagonal; long-range contacts are sparse (mostly zero).
3. Contact frequency decays sharply with genomic distance.

By storing only the main and a fixed number of super-diagonals as columns of a band matrix (diagonal-major storage: diagonal k stored in column k), we drastically reduce memory usage while enabling random access to Hi-C contacts. Additionally, mask and mask_row_col arrays track invalid or masked contacts to support downstream analysis.

Operations on this band_hic_matrix are as simple as on a numpy.ndarray; users can ignore these storage details.

This class stores only the main diagonal and up to (diag_num - 1) super-diagonals, exploiting symmetry by mirroring values for lower-triangular access.

shape

Shape of the original full Hi-C contact matrix (bin_num, bin_num), regardless of internal band storage format.

Type
tuple of int

dtype

Data type of the matrix elements, compatible with numpy dtypes.

Type
data-type

diag_num

Number of diagonals stored.

Type
int

bin_num

Number of bins (rows/columns) of the Hi-C matrix.

Type
int

data

Array of shape (*bin_num*, *diag_num*) storing banded Hi-C data.

Type
ndarray

mask

Mask for individual invalid entries. Stored as a boolean ndarray of shape (*bin_num*, *diag_num*) with the same shape as *data*.

Type
ndarray of bool or None

mask_row_col

Mask for entire rows and corresponding columns, indicating invalid bins. Stored as a boolean ndarray of shape (*bin_num*,). For computational convenience, row/column masks are also applied to the *mask* array to track masked entries.

Type
ndarray of bool or None

default_value

Default value for out-of-band entries. Entries out of the banded region and not stored in the data array will be set to this value.

Type
scalar

Examples

```
>>> import bandhic as bh
>>> import numpy as np
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.shape
(4, 4)
```

__init__(*contacts: coo_array | coo_matrix | tuple | ndarray*, *diag_num: int = 1*, *mask_row_col: ndarray | None = None*, *mask: Tuple[ndarray, ndarray] | None = None*, *dtype: type | None = None*, *default_value: int | float = 0*, *band_data_input: bool = False*) → None

Initialize a band_hic_matrix instance.

Parameters

- **contacts** (*{coo_array, coo_matrix, tuple, ndarray}*) – Input Hi-C data in COO format, tuple (data, (row, col)), full square array, or banded stored ndarray. For non-symmetric full arrays, only the upper-triangular part is used and the matrix is symmetrized. Full square arrays are not recommended for large matrices due to memory constraints.
- **diag_num** (*int, optional*) – Number of diagonals to store. Must be >=1 and <= matrix dimension. Default is 1.
- **mask_row_col** (*ndarray of bool or indices, optional*) – Mask for invalid rows/columns. Can be specified as: - A boolean array of shape (bin_num,) indicating which rows/columns to mask. - A list of indices to mask. Defaults to None (no masking).
- **mask** (*ndarray pair of (row_indices, col_indices), optional*) – Mask for invalid matrix entries. Can be specified as: - A tuple of two ndarray (row_indices, col_indices) listing positions to mask. Defaults to None (no masking).
- **dtype** (*data-type, optional*) – Desired numpy dtype; defaults to 'contacts' data dtype; compatible with numpy dtypes.
- **default_value** (*scalar, optional*) – Default value for unstored out-of-band entries. Default is 0.
- **band_data_input** (*bool, optional*) – If True, contacts is treated as precomputed band storage. Default is False.

Raises

ValueError – If contacts type is invalid, diag_num out of range, or array shape invalid.

Examples

Initialize from a SciPy COO matrix: >>> import bandhic as bh >>> import numpy as np >>> from scipy.sparse import coo_matrix >>> coo = coo_matrix(([1, 2, 3], ([0, 1, 2],[0, 1, 2])), shape=(3,3)) >>> mat1 = bh.band_hic_matrix(coo, diag_num=2) >>> mat1.data.shape (3, 2)

Initialize from a tuple (data, (row, col)): >>> mat2 = bh.band_hic_matrix(([4, 5, 6], ([0, 1, 2],[2, 1, 0])), diag_num=1) >>> mat2.data.shape (3, 1)

Initialize from a full dense array, only upper-triangular part is stored, lower part is symmetrized: >>> arr = np.arange(16).reshape(4,4) >>> mat3 = bh.band_hic_matrix(arr, diag_num=3) >>> mat3.data.shape (4, 3)

Initialize with row/column mask, this masks entire rows and corresponding columns: >>> mask = np.array([True, False, False, True]) >>> mat4 = bh.band_hic_matrix(arr, diag_num=2, mask_row_col=mask) >>> mat4.mask_row_col array([True, False, False, True])

mask_row_col is also supported as a list of indices: >>> mat4 = bh.band_hic_matrix(arr, diag_num=2, mask_row_col=[0, 3]) >>> mat4.mask_row_col array([True, False, False, True])

Initialize from precomputed banded storage: >>> band = mat3.data.copy() >>> mat5 = bh.band_hic_matrix(band, band_data_input=True) >>> mat5.data.shape (4, 3)

Methods

__init__(contacts[, diag_num, mask_row_col, ...])	Initialize a band_hic_matrix instance.
absolute(self, *args, **kwargs)	Perform element-wise 'absolute' operation.
add(self, other, *args, **kwargs)	Perform element-wise 'add' operation with two inputs.
add_mask(row_idx, col_idx)	Add mask entries for specified indices.
add_mask_row_col(mask_row_col)	Mask entire rows and corresponding columns.
all([axis, banded_only])	Test whether all (or any) array elements along a given axis evaluate to True.
any([axis, banded_only])	Test whether all (or any) array elements along a given axis evaluate to True.
arccos(self, *args, **kwargs)	Perform element-wise 'arccos' operation.
arccosh(self, *args, **kwargs)	Perform element-wise 'arccosh' operation.
arcsin(self, *args, **kwargs)	Perform element-wise 'arcsin' operation.
arcsinh(self, *args, **kwargs)	Perform element-wise 'arcsinh' operation.
arctan(self, *args, **kwargs)	Perform element-wise 'arctan' operation.
arctan2(self, other, *args, **kwargs)	Perform element-wise 'arctan2' operation with two inputs.
arctanh(self, *args, **kwargs)	Perform element-wise 'arctanh' operation.
astype(dtype[, copy])	Cast data to new dtype.
bitwise_and(self, other, *args, **kwargs)	Perform element-wise 'bitwise_and' operation with two inputs.
bitwise_or(self, other, *args, **kwargs)	Perform element-wise 'bitwise_or' operation with two inputs.
bitwise_xor(self, other, *args, **kwargs)	Perform element-wise 'bitwise_xor' operation with two inputs.
cbrt(self, *args, **kwargs)	Perform element-wise 'cbrt' operation.
clear_mask()	Clear all masks (both entry-level and row/column-level).
clip(min_val, max_val)	Clip data values to given range.
conjugate(self, *args, **kwargs)	Perform element-wise 'conjugate' operation.
copy()	Deep copy the object.
cos(self, *args, **kwargs)	Perform element-wise 'cos' operation.
cosh(self, *args, **kwargs)	Perform element-wise 'cosh' operation.
count_in_band()	Count the number of valid entries in the in-band region.
count_in_band_masked()	Count the number of masked entries in the in-band region.
count_masked()	Count the number of masked entries in the banded matrix.
count_out_band()	Count the number of valid entries in the out-of-band region.
count_out_band_masked()	Count the number of masked entries in the out-of-band region.
count_unmasked()	Count the number of unmasked entries in the banded matrix.
deg2rad(self, *args, **kwargs)	Perform element-wise 'deg2rad' operation.
degrees(self, *args, **kwargs)	Perform element-wise 'degrees' operation.

continues on next page

Table 2 – continued from previous page

diag(k)	Retrieve the k-th diagonal from the matrix.
divide(self, other, *args, **kwargs)	Perform element-wise 'divide' operation with two inputs.
divmod(self, other, *args, **kwargs)	Perform element-wise 'divmod' operation with two inputs.
drop_mask()	Clear the current mask by entry-level, but retain the row/column mask.
drop_mask_row_col()	Clear the current row/column mask.
dump(filename)	Save the band_hic_matrix object to a file.
equal(self, other, *args, **kwargs)	Perform element-wise 'equal' operation with two inputs.
exp(self, *args, **kwargs)	Perform element-wise 'exp' operation.
exp2(self, *args, **kwargs)	Perform element-wise 'exp2' operation.
expm1(self, *args, **kwargs)	Perform element-wise 'expm1' operation.
extract_row(idx[, extract_out_of_band])	Extract stored, unmasked band values for a given row or column.
fabs(self, *args, **kwargs)	Perform element-wise 'fabs' operation.
filled([fill_value, copy])	Fill masked entries in data with default value.
float_power(self, other, *args, **kwargs)	Perform element-wise 'float_power' operation with two inputs.
floor_divide(self, other, *args, **kwargs)	Perform element-wise 'floor_divide' operation with two inputs.
fmod(self, other, *args, **kwargs)	Perform element-wise 'fmod' operation with two inputs.
gcd(self, other, *args, **kwargs)	Perform element-wise 'gcd' operation with two inputs.
get_mask()	Get current mask array.
get_mask_row_col()	Get current row/column mask.
get_values(row_idx, col_idx)	Retrieve values considering mask.
greater(self, other, *args, **kwargs)	Perform element-wise 'greater' operation with two inputs.
greater_equal(self, other, *args, **kwargs)	Perform element-wise 'greater_equal' operation with two inputs.
heaviside(self, other, *args, **kwargs)	Perform element-wise 'heaviside' operation with two inputs.
hypot(self, other, *args, **kwargs)	Perform element-wise 'hypot' operation with two inputs.
init_mask()	Initialize mask for invalid entries based on matrix shape.
invert(self, *args, **kwargs)	Perform element-wise 'invert' operation.
itercols()	Iterate over the columns of the band_hic_matrix object.
iterrows()	Iterate over the rows of the band_hic_matrix object.
iterwindows(width[, step])	Iterate over the diagonals of the matrix with a specified window size.
lcm(self, other, *args, **kwargs)	Perform element-wise 'lcm' operation with two inputs.
left_shift(self, other, *args, **kwargs)	Perform element-wise 'left_shift' operation with two inputs.
less(self, other, *args, **kwargs)	Perform element-wise 'less' operation with two inputs.

continues on next page

Table 2 – continued from previous page

less_equal(self, other, *args, **kwargs)	Perform element-wise 'less_equal' operation with two inputs.
log(self, *args, **kwargs)	Perform element-wise 'log' operation.
log10(self, *args, **kwargs)	Perform element-wise 'log10' operation.
log1p(self, *args, **kwargs)	Perform element-wise 'log1p' operation.
log2(self, *args, **kwargs)	Perform element-wise 'log2' operation.
logaddexp(self, other, *args, **kwargs)	Perform element-wise 'logaddexp' operation with two inputs.
logaddexp2(self, other, *args, **kwargs)	Perform element-wise 'logaddexp2' operation with two inputs.
logical_and(self, other, *args, **kwargs)	Perform element-wise 'logical_and' operation with two inputs.
logical_or(self, other, *args, **kwargs)	Perform element-wise 'logical_or' operation with two inputs.
logical_xor(self, other, *args, **kwargs)	Perform element-wise 'logical_xor' operation with two inputs.
max([axis])	Compute the maximum value in the matrix or along a given axis.
maximum(self, other, *args, **kwargs)	Perform element-wise 'maximum' operation with two inputs.
mean([axis])	Compute the mean value of the matrix or along a given axis.
memory_usage()	Compute memory usage of *band_hic_matirx* object.
min([axis])	Compute the minimum value in the matrix or along a given axis.
minimum(self, other, *args, **kwargs)	Perform element-wise 'minimum' operation with two inputs.
multiply(self, other, *args, **kwargs)	Perform element-wise 'multiply' operation with two inputs.
negative(self, *args, **kwargs)	Perform element-wise 'negative' operation.
normalize([inplace])	Normalize each diagonal of the matrix to have zero mean and unit variance.
not_equal(self, other, *args, **kwargs)	Perform element-wise 'not_equal' operation with two inputs.
positive(self, *args, **kwargs)	Perform element-wise 'positive' operation.
power(self, other, *args, **kwargs)	Perform element-wise 'power' operation with two inputs.
prod([axis])	Compute the product of the values in the matrix or along a given axis.
ptp([axis])	Compute the peak-to-peak (maximum - minimum) value of the matrix or along a given axis.
rad2deg(self, *args, **kwargs)	Perform element-wise 'rad2deg' operation.
radians(self, *args, **kwargs)	Perform element-wise 'radians' operation.
reciprocal(self, *args, **kwargs)	Perform element-wise 'reciprocal' operation.
remainder(self, other, *args, **kwargs)	Perform element-wise 'remainder' operation with two inputs.
right_shift(self, other, *args, **kwargs)	Perform element-wise 'right_shift' operation with two inputs.
rint(self, *args, **kwargs)	Perform element-wise 'rint' operation.
set_diag(k, values)	Set values in the k-th diagonal of the matrix.
set_values(row_idx, col_idx, values)	Set values at specified row and column indices.
sign(self, *args, **kwargs)	Perform element-wise 'sign' operation.

continues on next page

Table 2 – continued from previous page

sin(self, *args, **kwargs)	Perform element-wise 'sin' operation.
sinh(self, *args, **kwargs)	Perform element-wise 'sinh' operation.
sqrt(self, *args, **kwargs)	Perform element-wise 'sqrt' operation.
square(self, *args, **kwargs)	Perform element-wise 'square' operation.
std([axis])	Compute the standard deviation of the values in the matrix or along a given axis.
subtract(self, other, *args, **kwargs)	Perform element-wise 'subtract' operation with two inputs.
sum([axis])	Compute the sum of the values in the matrix or along a given axis.
tan(self, *args, **kwargs)	Perform element-wise 'tan' operation.
tanh(self, *args, **kwargs)	Perform element-wise 'tanh' operation.
tocoo([drop_zeros])	Convert the matrix to COO format.
tocsr()	Convert the matrix to CSR format.
todense()	Convert the band matrix to a dense format.
unmask([indices])	Remove mask entries for specified indices or clear all.
var([axis])	Compute the variance of the values in the matrix or along a given axis.

4.2 Detailed Class Documentation

class bandhic.**band_hic_matrix**(*contacts: coo_array | coo_matrix | tuple | ndarray*, *diag_num: int = 1*, *mask_row_col: ndarray | None = None*, *mask: Tuple[ndarray, ndarray] | None = None*, *dtype: type | None = None*, *default_value: int | float = 0*, *band_data_input: bool = False*)

Bases: `NDArrayOperatorsMixin`

Symmetric banded matrix stored in upper-triangular format. This storage format is motivated by high-resolution Hi-C data characteristics:

1. Symmetry of contact maps.
2. Interaction frequency concentrated near the diagonal; long-range contacts are sparse (mostly zero).
3. Contact frequency decays sharply with genomic distance.

By storing only the main and a fixed number of super-diagonals as columns of a band matrix (diagonal-major storage: diagonal k stored in column k), we drastically reduce memory usage while enabling random access to Hi-C contacts. Additionally, mask and mask_row_col arrays track invalid or masked contacts to support downstream analysis.

Operations on this band_hic_matrix are as simple as on a numpy.ndarray; users can ignore these storage details.

This class stores only the main diagonal and up to (diag_num - 1) super-diagonals, exploiting symmetry by mirroring values for lower-triangular access.

shape

Shape of the original full Hi-C contact matrix (bin_num, bin_num), regardless of internal band storage format.

Type
tuple of int

dtype

Data type of the matrix elements, compatible with numpy dtypes.

Type
data-type

diag_num

Number of diagonals stored.

Type
int

bin_num

Number of bins (rows/columns) of the Hi-C matrix.

Type
int

data

Array of shape (*bin_num*, *diag_num*) storing banded Hi-C data.

Type
ndarray

mask

Mask for individual invalid entries. Stored as a boolean ndarray of shape (*bin_num*, *diag_num*) with the same shape as *data*.

Type
ndarray of bool or None

mask_row_col

Mask for entire rows and corresponding columns, indicating invalid bins. Stored as a boolean ndarray of shape (*bin_num*,). For computational convenience, row/column masks are also applied to the *mask* array to track masked entries.

Type
ndarray of bool or None

default_value

Default value for out-of-band entries. Entries out of the banded region and not stored in the data array will be set to this value.

Type
scalar

Examples

```
>>> import bandhic as bh
>>> import numpy as np
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.shape
(4, 4)
```

__init__(*contacts: coo_array | coo_matrix | tuple | ndarray*, *diag_num: int = 1*, *mask_row_col: ndarray | None = None*, *mask: Tuple[ndarray, ndarray] | None = None*, *dtype: type | None = None*, *default_value: int | float = 0*, *band_data_input: bool = False*) → None

Initialize a band_hic_matrix instance.

Parameters

- **contacts** (*{coo_array, coo_matrix, tuple, ndarray}*) – Input Hi-C data in COO format, tuple (data, (row, col)), full square array, or banded stored ndarray. For non-symmetric full arrays, only the upper-triangular part is used and the matrix is symmetrized. Full square arrays are not recommended for large matrices due to memory constraints.
- **diag_num** (*int, optional*) – Number of diagonals to store. Must be >=1 and <= matrix dimension. Default is 1.
- **mask_row_col** (*ndarray of bool or indices, optional*) – Mask for invalid rows/columns. Can be specified as: - A boolean array of shape (bin_num,) indicating which rows/columns to mask. - A list of indices to mask. Defaults to None (no masking).
- **mask** (*ndarray pair of (row_indices, col_indices), optional*) – Mask for invalid matrix entries. Can be specified as: - A tuple of two ndarray (row_indices, col_indices) listing positions to mask. Defaults to None (no masking).
- **dtype** (*data-type, optional*) – Desired numpy dtype; defaults to 'contacts' data dtype; compatible with numpy dtypes.
- **default_value** (*scalar, optional*) – Default value for unstored out-of-band entries. Default is 0.
- **band_data_input** (*bool, optional*) – If True, contacts is treated as precomputed band storage. Default is False.

Raises

ValueError – If contacts type is invalid, diag_num out of range, or array shape invalid.

Examples

Initialize from a SciPy COO matrix: >>> import bandhic as bh >>> import numpy as np >>> from scipy.sparse import coo_matrix >>> coo = coo_matrix(([1, 2, 3], ([0, 1, 2],[0, 1, 2])), shape=(3,3)) >>> mat1 = bh.band_hic_matrix(coo, diag_num=2) >>> mat1.data.shape (3, 2)

Initialize from a tuple (data, (row, col)): >>> mat2 = bh.band_hic_matrix(([4, 5, 6], ([0, 1, 2],[2, 1, 0])), diag_num=1) >>> mat2.data.shape (3, 1)

Initialize from a full dense array, only upper-triangular part is stored, lower part is symmetrized: >>> arr = np.arange(16).reshape(4,4) >>> mat3 = bh.band_hic_matrix(arr, diag_num=3) >>> mat3.data.shape (4, 3)

Initialize with row/column mask, this masks entire rows and corresponding columns: >>> mask = np.array([True, False, False, True]) >>> mat4 = bh.band_hic_matrix(arr, diag_num=2, mask_row_col=mask) >>> mat4.mask_row_col array([True, False, False, True])

mask_row_col is also supported as a list of indices: >>> mat4 = bh.band_hic_matrix(arr, diag_num=2, mask_row_col=[0, 3]) >>> mat4.mask_row_col array([True, False, False, True])

Initialize from precomputed banded storage: >>> band = mat3.data.copy() >>> mat5 = bh.band_hic_matrix(band, band_data_input=True) >>> mat5.data.shape (4, 3)

absolute(*self*, **args*, ***kwargs*)

Perform element-wise 'absolute' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.absolute.
- ****kwargs** (*dict*) – Keyword arguments for numpy.absolute.

Returns

Result of element-wise 'absolute' operation.

Return type
band_hic_matrix

See also

`numpy.absolute`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.absolute()
```

add(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'add' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.add.
- ****kwargs** (*dict*) – Keyword arguments for numpy.add.

Returns
Result of element-wise 'add' operation.

Return type
band_hic_matrix

See also

`numpy.add`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.add(other)
```

add_mask(*row_idx: Sequence[int] | ndarray*, *col_idx: Sequence[int] | ndarray*) → None

Add mask entries for specified indices.

Parameters

- **row_idx** (*array-like of int*) – Row indices to mask.
- **col_idx** (*array-like of int*) – Column indices to mask.

Raises
ValueError – If row_idx and col_idx have different shapes.

Examples

```
>>> import bandhic as bh
>>> import numpy as np
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.add_mask([0, 1], [1, 2])
```

add_mask_row_col(*mask_row_col: Sequence[int] | Sequence[bool] | ndarray | int*) → None

Mask entire rows and corresponding columns.

Parameters

mask_row_col (*array-like of int or bool*) – If boolean array of shape (bin_num,), True entries indicate rows/columns to mask. If integer array or sequence, treated as indices of rows/columns to mask.

Raises

ValueError – If integer indices are out of bounds.

Examples

```
>>> import bandhic as bh
>>> mask = np.array([True, False, False])
>>> mat = bh.band_hic_matrix(np.eye(3), diag_num=2)
>>> mat.add_mask_row_col(mask)
```

all(*axis: int | str | None = None*, *banded_only: bool = False*) → bool_ | ndarray[bool_, Any]

Test whether all (or any) array elements along a given axis evaluate to True.

Parameters

- **axis** (*None, int, or {'row','col','diag'}, optional*) – Axis along which to test.
- **banded_only** (*bool, optional*) – If True, only consider stored band elements; ignore out-of-band values. Default is False.

Returns

Boolean result(s) of the test.

Return type

bool or ndarray

Raises

ValueError – If axis is not supported.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(3), diag_num=2)
>>> mat.all()
False
>>> mat.any(axis='diag', banded_only=True)
array([ True, False])
```

any(*axis: int | str | None = None*, *banded_only: bool = False*) → bool | ndarray

Test whether all (or any) array elements along a given axis evaluate to True.

Parameters

- **axis** (*None, int, or {'row','col','diag'}, optional*) – Axis along which to test.
- **banded_only** (*bool, optional*) – If True, only consider stored band elements; ignore out-of-band values. Default is False.

Returns
Boolean result(s) of the test.

Return type
bool or ndarray

Raises
ValueError – If axis is not supported.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(3), diag_num=2)
>>> mat.all()
False
>>> mat.any(axis='diag', banded_only=True)
array([ True, False])
```

arccos(*self*, **args*, ***kwargs*)

Perform element-wise 'arccos' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arccos.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arccos.

Returns
Result of element-wise 'arccos' operation.

Return type
band_hic_matrix

See also

numpy.arccos

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arccos()
```

arccosh(*self*, **args*, ***kwargs*)

Perform element-wise 'arccosh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arccosh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arccosh.

Returns
Result of element-wise 'arccosh' operation.

Return type
band_hic_matrix

See also

numpy.arccosh

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arccosh()
```

arcsin(*self*, **args*, ***kwargs*)

Perform element-wise 'arcsin' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arcsin.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arcsin.

Returns
Result of element-wise 'arcsin' operation.

Return type
band_hic_matrix

See also

numpy.arcsin

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arcsin()
```

arcsinh(*self*, **args*, ***kwargs*)

Perform element-wise 'arcsinh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arcsinh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arcsinh.

Returns
Result of element-wise 'arcsinh' operation.

Return type
band_hic_matrix

See also

`numpy.arcsinh`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arcsinh()
```

arctan(*self*, **args*, ***kwargs*)

Perform element-wise 'arctan' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arctan.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arctan.

Returns
Result of element-wise 'arctan' operation.

Return type
band_hic_matrix

See also

`numpy.arctan`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arctan()
```

arctan2(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'arctan2' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.arctan2.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arctan2.

Returns
Result of element-wise 'arctan2' operation.

Return type
band_hic_matrix

> **See also**
>
> numpy.arctan2

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.arctan2(other)
```

arctanh(*self*, **args*, ***kwargs*)

Perform element-wise 'arctanh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.arctanh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.arctanh.

Returns

Result of element-wise 'arctanh' operation.

Return type

band_hic_matrix

> **See also**
>
> numpy.arctanh

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.arctanh()
```

astype(*dtype: type*, *copy: bool = False*) → *band_hic_matrix*

Cast data to new dtype.

Parameters

- **type** (*data-type*) – Target dtype.
- **copy** (*bool*) – If True, the operation is performed in place. Default is False.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat = mat.astype(np.float32)
```

bitwise_and(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'bitwise_and' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.bitwise_and.
- ****kwargs** (*dict*) – Keyword arguments for numpy.bitwise_and.

Returns
Result of element-wise 'bitwise_and' operation.

Return type
band_hic_matrix

See also

numpy.bitwise_and

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.bitwise_and(other)
```

bitwise_or(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'bitwise_or' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.bitwise_or.
- ****kwargs** (*dict*) – Keyword arguments for numpy.bitwise_or.

Returns
Result of element-wise 'bitwise_or' operation.

Return type
band_hic_matrix

See also

numpy.bitwise_or

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.bitwise_or(other)
```

bitwise_xor(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'bitwise_xor' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.bitwise_xor.
- ****kwargs** (*dict*) – Keyword arguments for numpy.bitwise_xor.

Returns

Result of element-wise 'bitwise_xor' operation.

Return type

band_hic_matrix

See also

numpy.bitwise_xor

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.bitwise_xor(other)
```

cbrt(*self*, **args*, ***kwargs*)

Perform element-wise 'cbrt' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.cbrt.
- ****kwargs** (*dict*) – Keyword arguments for numpy.cbrt.

Returns

Result of element-wise 'cbrt' operation.

Return type

band_hic_matrix

See also

numpy.cbrt

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.cbrt()
```

clear_mask() → None

Clear all masks (both entry-level and row/column-level).

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.clear_mask()
```

clip(*min_val: Number*, *max_val: Number*) → None

Clip data values to given range.

Parameters

- **min_val** (*scalar*)
- **max_val** (*scalar*)

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat = mat.clip(0, 10)
```

conjugate(*self*, **args*, ***kwargs*)

Perform element-wise 'conjugate' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.conjugate.
- ****kwargs** (*dict*) – Keyword arguments for numpy.conjugate.

Returns

Result of element-wise 'conjugate' operation.

Return type

band_hic_matrix

See also

numpy.conjugate

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.conjugate()
```

copy() → *band_hic_matrix*

Deep copy the object.

Returns

A deep copy.

Return type
band_hic_matrix

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat2 = mat.copy()
```

cos(*self*, **args*, ***kwargs*)

Perform element-wise 'cos' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.cos.
- ****kwargs** (*dict*) – Keyword arguments for numpy.cos.

Returns
Result of element-wise 'cos' operation.

Return type
band_hic_matrix

See also

numpy.cos

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.cos()
```

cosh(*self*, **args*, ***kwargs*)

Perform element-wise 'cosh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.cosh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.cosh.

Returns
Result of element-wise 'cosh' operation.

Return type
band_hic_matrix

See also

numpy.cosh

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.cosh()
```

count_in_band()

Count the number of valid entries in the in-band region.

Returns
Number of valid entries in the in-band region.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_in_band()
10
```

count_in_band_masked()

Count the number of masked entries in the in-band region.

Returns
Number of masked entries in the in-band region.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_in_band_masked()
0
```

count_masked()

Count the number of masked entries in the banded matrix. This counts the number of entries in the upper triangular part of the matrix, excluding the diagonal and valid entries. :returns: Number of valid entries in the banded matrix.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_masked()
0
```

Return type
int

count_out_band()

Count the number of valid entries in the out-of-band region.

Returns
Number of valid entries in the out-of-band region.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_out_band()
6
```

count_out_band_masked()

Count the number of masked entries in the out-of-band region.

Returns
Number of masked entries in the out-of-band region.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_out_band_masked()
0
```

count_unmasked()

Count the number of unmasked entries in the banded matrix.

Returns
Number of unmasked entries in the banded matrix.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.count_unmasked()
16
```

deg2rad(*self*, **args*, ***kwargs*)

Perform element-wise 'deg2rad' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.deg2rad.
- ****kwargs** (*dict*) – Keyword arguments for numpy.deg2rad.

Returns
Result of element-wise 'deg2rad' operation.

Return type
band_hic_matrix

See also

`numpy.deg2rad`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.deg2rad()
```

degrees(*self*, **args*, ***kwargs*)

Perform element-wise 'degrees' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.degrees.
- ****kwargs** (*dict*) – Keyword arguments for numpy.degrees.

Returns
Result of element-wise 'degrees' operation.

Return type
band_hic_matrix

See also

`numpy.degrees`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.degrees()
```

diag(*k: int*) → ndarray | MaskedArray

Retrieve the k-th diagonal from the matrix.

Parameters
k (*int*) – The diagonal index to retrieve.

Returns
The k-th diagonal values; masked if mask is set.

Return type
ndarray or MaskedArray

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat.diag(1)
array([1., 1.])
```

divide(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'divide' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.divide.
- ****kwargs** (*dict*) – Keyword arguments for numpy.divide.

Returns

Result of element-wise 'divide' operation.

Return type

band_hic_matrix

See also

numpy.divide

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.divide(other)
```

divmod(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'divmod' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.divmod.
- ****kwargs** (*dict*) – Keyword arguments for numpy.divmod.

Returns

Result of element-wise 'divmod' operation.

Return type

band_hic_matrix

See also

numpy.divmod

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.divmod(other)
```

drop_mask() → None

Clear the current mask by entry-level, but retain the row/column mask.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2, mask=(np.array([[0, 1], [1,␣
↪2]])))
>>> mat.drop_mask()
```

Notes

This method is useful when you want to keep the row/column mask but clear the entry-level mask. It will not affect the row/column mask, allowing you to maintain the masking of entire rows and columns. >>> mat.mask

drop_mask_row_col()

Clear the current row/column mask.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.drop_mask_row_col()
```

dump(*filename: str*) → None

Save the band_hic_matrix object to a file.

Parameters

filename (*str*) – The name of the file to save the object to.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat.dump('myfile.npz')
```

equal(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'equal' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.equal.
- ****kwargs** (*dict*) – Keyword arguments for numpy.equal.

Returns
Result of element-wise 'equal' operation.

Return type
band_hic_matrix

See also

`numpy.equal`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.equal(other)
```

exp(*self*, **args*, ***kwargs*)

Perform element-wise 'exp' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.exp.
- ****kwargs** (*dict*) – Keyword arguments for numpy.exp.

Returns
Result of element-wise 'exp' operation.

Return type
band_hic_matrix

See also

`numpy.exp`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.exp()
```

exp2(*self*, **args*, ***kwargs*)

Perform element-wise 'exp2' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.exp2.
- ****kwargs** (*dict*) – Keyword arguments for numpy.exp2.

Returns
Result of element-wise 'exp2' operation.

Return type
band_hic_matrix

See also

`numpy.exp2`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.exp2()
```

expm1(*self*, **args*, ***kwargs*)

Perform element-wise 'expm1' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.expm1.
- ****kwargs** (*dict*) – Keyword arguments for numpy.expm1.

Returns
Result of element-wise 'expm1' operation.

Return type
band_hic_matrix

See also

`numpy.expm1`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.expm1()
```

extract_row(*idx: int*, *extract_out_of_band: bool = True*) → ndarray | MaskedArray

Extract stored, unmasked band values for a given row or column.

Parameters

- **idx** (*int*) – Row (or column, due to symmetry) index for which to extract band values.
- **extract_out_of_band** (*bool, optional*) – If True, include out-of-band entries filled with *default_value* and masked if appropriate. If False (default), return only the stored band values.

Returns
If *extract_out_of_band=False*, a 1D array of length up to *diag_num* containing band values. If *extract_out_of_band=True*, a 1D array of length *bin_num* with all row/column values.

Return type
ndarray or MaskedArray

Raises
ValueError – If *idx* is outside the range [0, bin_num-1].

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.extract_row(0, extract_out_of_band=False)
array([0, 1])
>>> mat.extract_row(0)
array([0, 1, 0])
```

fabs(*self*, **args*, ***kwargs*)

Perform element-wise 'fabs' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.fabs.
- ****kwargs** (*dict*) – Keyword arguments for numpy.fabs.

Returns
Result of element-wise 'fabs' operation.

Return type
band_hic_matrix

See also

numpy.fabs

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.fabs()
```

filled(*fill_value: int | float | None = None*, *copy: bool = True*) → *band_hic_matrix*

Fill masked entries in data with default value.

Parameters
fill_value (*scalar*) – Value to assign to masked entries.

Raises
ValueError – If no mask is initialized.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2, mask = ([0,1],[1,2]))
>>> mat.filled()
    band_hic_matrix(shape=(4, 4), diag_num=2, dtype=float64)
```

float_power(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'float_power' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.float_power.
- ****kwargs** (*dict*) – Keyword arguments for numpy.float_power.

Returns

Result of element-wise 'float_power' operation.

Return type

band_hic_matrix

See also

numpy.float_power

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.float_power(other)
```

floor_divide(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'floor_divide' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.floor_divide.
- ****kwargs** (*dict*) – Keyword arguments for numpy.floor_divide.

Returns

Result of element-wise 'floor_divide' operation.

Return type

band_hic_matrix

See also

numpy.floor_divide

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
```

(continues on next page)

(continued from previous page)

```
>>> other = mat.copy()
>>> result = mat.floor_divide(other)
```

fmod(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'fmod' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.fmod.
- ****kwargs** (*dict*) – Keyword arguments for numpy.fmod.

Returns

Result of element-wise 'fmod' operation.

Return type

band_hic_matrix

See also

numpy.fmod

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.fmod(other)
```

gcd(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'gcd' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.gcd.
- ****kwargs** (*dict*) – Keyword arguments for numpy.gcd.

Returns

Result of element-wise 'gcd' operation.

Return type

band_hic_matrix

See also

numpy.gcd

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.gcd(other)
```

get_mask()

Get current mask array.

Returns
Current mask array or None if no mask.

Return type
ndarray or None

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mask = mat.get_mask()
```

get_mask_row_col()

Get current row/column mask.

Returns
Current row/column mask or None if no mask.

Return type
ndarray or None

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mask_row_col = mat.get_mask_row_col()
```

get_values(*row_idx: ndarray | Iterable[int]*, *col_idx: ndarray | Iterable[int]*) → ndarray | MaskedArray

Retrieve values considering mask.

Parameters

- **row_idx** (*array-like of int*) – Row indices.
- **col_idx** (*array-like of int*) – Column indices.

Returns
Retrieved values; masked entries yield masked results.

Return type
ndarray or MaskedArray

Raises
ValueError – If indices exceed matrix dimensions.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat.get_values([0,1],[1,2])
array([1., 1.])
```

greater(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'greater' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.greater.
- ****kwargs** (*dict*) – Keyword arguments for numpy.greater.

Returns

Result of element-wise 'greater' operation.

Return type

band_hic_matrix

See also

numpy.greater

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.greater(other)
```

greater_equal(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'greater_equal' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.greater_equal.
- ****kwargs** (*dict*) – Keyword arguments for numpy.greater_equal.

Returns

Result of element-wise 'greater_equal' operation.

Return type

band_hic_matrix

See also

numpy.greater_equal

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.greater_equal(other)
```

heaviside(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'heaviside' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.heaviside.
- ****kwargs** (*dict*) – Keyword arguments for numpy.heaviside.

Returns

Result of element-wise 'heaviside' operation.

Return type

band_hic_matrix

See also

numpy.heaviside

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.heaviside(other)
```

hypot(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'hypot' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.hypot.
- ****kwargs** (*dict*) – Keyword arguments for numpy.hypot.

Returns

Result of element-wise 'hypot' operation.

Return type

band_hic_matrix

See also

numpy.hypot

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.hypot(other)
```

init_mask()

Initialize mask for invalid entries based on matrix shape.

Raises

ValueError – If mask is already initialized.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(4), diag_num=2)
>>> mat.init_mask()
```

invert(*self*, **args*, ***kwargs*)

Perform element-wise 'invert' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.invert.
- ****kwargs** (*dict*) – Keyword arguments for numpy.invert.

Returns

Result of element-wise 'invert' operation.

Return type

band_hic_matrix

See also

numpy.invert

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.invert()
```

itercols() → Iterator[ndarray]

Iterate over the columns of the band_hic_matrix object.

Yields

ndarray – The values in the current column.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> for col in mat.itercols():
...     print(col)
[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]
```

iterrows() → Iterator[ndarray]

Iterate over the rows of the band_hic_matrix object.

Yields

ndarray – The values in the current row.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> for row in mat.iterrows():
...     print(row)
[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]
```

iterwindows(*width: int*, *step: int = 1*) → Iterator[*band_hic_matrix*]

Iterate over the diagonals of the matrix with a specified window size.

Parameters

- **width** (*int*) – The size of the window to iterate over.
- **step** (*int, optional*) – Step size between windows. Default is 1.

Yields

band_hic_matrix – The values in the current window.

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> for win in mat.iterwindows(2):
...     print(win)
band_hic_matrix(shape=(2, 2), diag_num=2, dtype=<class 'numpy.float64'>)
band_hic_matrix(shape=(2, 2), diag_num=2, dtype=<class 'numpy.float64'>)
```

lcm(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'lcm' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.lcm.

- ****kwargs** (*dict*) – Keyword arguments for numpy.lcm.

Returns
Result of element-wise 'lcm' operation.

Return type
band_hic_matrix

See also

numpy.lcm

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.lcm(other)
```

left_shift(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'left_shift' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.left_shift.
- ****kwargs** (*dict*) – Keyword arguments for numpy.left_shift.

Returns
Result of element-wise 'left_shift' operation.

Return type
band_hic_matrix

See also

numpy.left_shift

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.left_shift(other)
```

less(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'less' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.

- ***args** (*tuple*) – Additional positional arguments for numpy.less.
- ****kwargs** (*dict*) – Keyword arguments for numpy.less.

Returns
Result of element-wise 'less' operation.

Return type
band_hic_matrix

See also

`numpy.less`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.less(other)
```

less_equal(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'less_equal' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.less_equal.
- ****kwargs** (*dict*) – Keyword arguments for numpy.less_equal.

Returns
Result of element-wise 'less_equal' operation.

Return type
band_hic_matrix

See also

`numpy.less_equal`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.less_equal(other)
```

log(*self*, **args*, ***kwargs*)

Perform element-wise 'log' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.

- ***args** (*tuple*) – Additional positional arguments for numpy.log.
- ****kwargs** (*dict*) – Keyword arguments for numpy.log.

Returns
Result of element-wise 'log' operation.

Return type
band_hic_matrix

See also

`numpy.log`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.log()
```

log10(*self*, **args*, ***kwargs*)

Perform element-wise 'log10' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.log10.
- ****kwargs** (*dict*) – Keyword arguments for numpy.log10.

Returns
Result of element-wise 'log10' operation.

Return type
band_hic_matrix

See also

`numpy.log10`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.log10()
```

log1p(*self*, **args*, ***kwargs*)

Perform element-wise 'log1p' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.log1p.
- ****kwargs** (*dict*) – Keyword arguments for numpy.log1p.

Returns
Result of element-wise 'log1p' operation.

Return type
band_hic_matrix

See also

numpy.log1p

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.log1p()
```

log2(*self*, **args*, ***kwargs*)

Perform element-wise 'log2' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.log2.
- ****kwargs** (*dict*) – Keyword arguments for numpy.log2.

Returns
Result of element-wise 'log2' operation.

Return type
band_hic_matrix

See also

numpy.log2

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.log2()
```

logaddexp(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'logaddexp' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.logaddexp.
- ****kwargs** (*dict*) – Keyword arguments for numpy.logaddexp.

Returns
Result of element-wise 'logaddexp' operation.

Return type
band_hic_matrix

See also

numpy.logaddexp

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.logaddexp(other)
```

logaddexp2(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'logaddexp2' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.logaddexp2.
- ****kwargs** (*dict*) – Keyword arguments for numpy.logaddexp2.

Returns
Result of element-wise 'logaddexp2' operation.

Return type
band_hic_matrix

See also

numpy.logaddexp2

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.logaddexp2(other)
```

logical_and(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'logical_and' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.logical_and.
- ****kwargs** (*dict*) – Keyword arguments for numpy.logical_and.

Returns
Result of element-wise 'logical_and' operation.

Return type
band_hic_matrix

See also

numpy.logical_and

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.logical_and(other)
```

logical_or(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'logical_or' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.logical_or.
- ****kwargs** (*dict*) – Keyword arguments for numpy.logical_or.

Returns
Result of element-wise 'logical_or' operation.

Return type
band_hic_matrix

See also

numpy.logical_or

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.logical_or(other)
```

logical_xor(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'logical_xor' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.logical_xor.

- ****kwargs** (*dict*) – Keyword arguments for numpy.logical_xor.

Returns
Result of element-wise 'logical_xor' operation.

Return type
band_hic_matrix

See also

numpy.logical_xor

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.logical_xor(other)
```

max(*axis: int | str | None = None*) → Number | ndarray

Compute the maximum value in the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the maximum: - None: compute over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Maximum value(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.max()  # global maximum
8
>>> mat.max(axis='row')
array([1, 5, 8])
```

maximum(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'maximum' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.maximum.
- ****kwargs** (*dict*) – Keyword arguments for numpy.maximum.

Returns
Result of element-wise 'maximum' operation.

Return type
band_hic_matrix

See also

`numpy.maximum`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.maximum(other)
```

mean(*axis: int | str | None = None*) → Number | ndarray

Compute the mean value of the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the mean: - None: mean over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Mean value(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.mean()  # mean of all elements
2.6666666666666665
>>> mat.mean(axis='diag')
array([4., 3.])
```

memory_usage() → int

Compute memory usage of *band_hic_matirx* object.

Returns
Size in bytes.

Return type
int

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> mat.memory_usage()
772
```

min(*axis: int | str | None = None*) → Number | ndarray

Compute the minimum value in the matrix or along a given axis.

Parameters

axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the minimum: - None: compute over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns

Minimum value(s) along the specified axis.

Return type

scalar or ndarray

Raises

ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.min()  # global minimum
0
>>> mat.min(axis='row')
array([0, 1, 0])
```

minimum(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'minimum' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.minimum.
- ****kwargs** (*dict*) – Keyword arguments for numpy.minimum.

Returns

Result of element-wise 'minimum' operation.

Return type

band_hic_matrix

See also

numpy.minimum

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.minimum(other)
```

multiply(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'multiply' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.multiply.
- ****kwargs** (*dict*) – Keyword arguments for numpy.multiply.

Returns

Result of element-wise 'multiply' operation.

Return type

band_hic_matrix

See also

numpy.multiply

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.multiply(other)
```

negative(*self*, **args*, ***kwargs*)

Perform element-wise 'negative' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.negative.
- ****kwargs** (*dict*) – Keyword arguments for numpy.negative.

Returns

Result of element-wise 'negative' operation.

Return type

band_hic_matrix

See also

numpy.negative

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.negative()
```

normalize(*inplace: bool = False*) → None

Normalize each diagonal of the matrix to have zero mean and unit variance. This modifies the matrix in place. :raises UserWarning: If any diagonal has zero standard deviation, it will be set to zero.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat = mat.normalize()
```

not_equal(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'not_equal' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.not_equal.
- ****kwargs** (*dict*) – Keyword arguments for numpy.not_equal.

Returns

Result of element-wise 'not_equal' operation.

Return type

band_hic_matrix

See also

numpy.not_equal

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.not_equal(other)
```

positive(*self*, **args*, ***kwargs*)

Perform element-wise 'positive' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.positive.
- ****kwargs** (*dict*) – Keyword arguments for numpy.positive.

Returns

Result of element-wise 'positive' operation.

Return type
band_hic_matrix

See also

numpy.positive

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.positive()
```

power(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'power' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.power.
- ****kwargs** (*dict*) – Keyword arguments for numpy.power.

Returns
Result of element-wise 'power' operation.

Return type
band_hic_matrix

See also

numpy.power

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.power(other)
```

prod(*axis: int | str | None = None*) → Number | ndarray

Compute the product of the values in the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the product: - None: product over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Product(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(1, 10).reshape(3,3), diag_num=2)
>>> mat.prod()  # product of all elements
0
>>> mat.prod(axis='row')
array([ 0, 60, 0])
```

ptp(*axis: int | str | None = None*) → Number | ndarray

Compute the peak-to-peak (maximum - minimum) value of the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the peak-to-peak value: - None: ptp over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Peak-to-peak value(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.ptp()  # ptp of all elements
8
>>> mat.ptp(axis='row')
array([1, 4, 8])
```

rad2deg(*self*, **args*, ***kwargs*)

Perform element-wise 'rad2deg' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.rad2deg.
- ****kwargs** (*dict*) – Keyword arguments for numpy.rad2deg.

Returns
Result of element-wise 'rad2deg' operation.

Return type
band_hic_matrix

See also

numpy.rad2deg

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.rad2deg()
```

radians(*self*, **args*, ***kwargs*)

Perform element-wise 'radians' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.radians.
- ****kwargs** (*dict*) – Keyword arguments for numpy.radians.

Returns

Result of element-wise 'radians' operation.

Return type

band_hic_matrix

See also

numpy.radians

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.radians()
```

reciprocal(*self*, **args*, ***kwargs*)

Perform element-wise 'reciprocal' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.reciprocal.
- ****kwargs** (*dict*) – Keyword arguments for numpy.reciprocal.

Returns

Result of element-wise 'reciprocal' operation.

Return type

band_hic_matrix

See also

numpy.reciprocal

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.reciprocal()
```

remainder(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'remainder' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.remainder.
- ****kwargs** (*dict*) – Keyword arguments for numpy.remainder.

Returns

Result of element-wise 'remainder' operation.

Return type

band_hic_matrix

See also

numpy.remainder

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.remainder(other)
```

right_shift(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'right_shift' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.right_shift.
- ****kwargs** (*dict*) – Keyword arguments for numpy.right_shift.

Returns

Result of element-wise 'right_shift' operation.

Return type

band_hic_matrix

See also

numpy.right_shift

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.right_shift(other)
```

rint(*self*, **args*, ***kwargs*)

Perform element-wise 'rint' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.rint.
- ****kwargs** (*dict*) – Keyword arguments for numpy.rint.

Returns

Result of element-wise 'rint' operation.

Return type

band_hic_matrix

See also

numpy.rint

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.rint()
```

set_diag(*k: int*, *values: Iterable[Any]*) → None

Set values in the k-th diagonal of the matrix.

Parameters

- **k** (*int*) – The diagonal index to set.
- **values** (*array-like*) – The values to set in the diagonal.

Raises

ValueError – If k is out of range or values length mismatch.

Examples

```
>>> import bandhic as bh
>>> mat = bh.zeros((4,4), diag_num=3)
>>> mat.set_diag(1, [9,9,9])
```

(continues on next page)

(continued from previous page)

```
>>> mat.diag(1)
array([9., 9., 9.])
```

set_values(*row_idx: ndarray*, *col_idx: ndarray*, *values: Number | ndarray*) → None

Set values at specified row and column indices.

Parameters

- **row_idx** (*array-like of int*) – Row indices where values will be set.
- **col_idx** (*array-like of int*) – Column indices where values will be set.
- **values** (*scalar or array-like*) – Values to assign.

Examples

```
>>> import bandhic as bh
>>> mat = bh.zeros((3,3), diag_num=2, dtype=int)
>>> mat.set_values([0,1], [1,2], [4,5])
>>> mat[0,1]
4
```

Notes

Writing to masked positions will update the underlying data but will not clear the mask.

sign(*self*, **args*, ***kwargs*)

Perform element-wise 'sign' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.sign.
- ****kwargs** (*dict*) – Keyword arguments for numpy.sign.

Returns

Result of element-wise 'sign' operation.

Return type

band_hic_matrix

See also

numpy.sign

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.sign()
```

sin(*self*, **args*, ***kwargs*)

Perform element-wise 'sin' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.sin.
- ****kwargs** (*dict*) – Keyword arguments for numpy.sin.

Returns
Result of element-wise 'sin' operation.

Return type
band_hic_matrix

See also

numpy.sin

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.sin()
```

sinh(*self*, **args*, ***kwargs*)

Perform element-wise 'sinh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.sinh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.sinh.

Returns
Result of element-wise 'sinh' operation.

Return type
band_hic_matrix

See also

numpy.sinh

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.sinh()
```

sqrt(*self*, **args*, ***kwargs*)

Perform element-wise 'sqrt' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.sqrt.
- ****kwargs** (*dict*) – Keyword arguments for numpy.sqrt.

Returns
Result of element-wise 'sqrt' operation.

Return type
band_hic_matrix

See also

`numpy.sqrt`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.sqrt()
```

square(*self*, **args*, ***kwargs*)

Perform element-wise 'square' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.square.
- ****kwargs** (*dict*) – Keyword arguments for numpy.square.

Returns
Result of element-wise 'square' operation.

Return type
band_hic_matrix

See also

`numpy.square`

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.square()
```

std(*axis: int | str | None = None*) → Number | ndarray

Compute the standard deviation of the values in the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the standard deviation: - None: std over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Standard deviation(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.std()  # std of all elements
2.748737083745107
>>> mat.std(axis='diag')
array([3.26598632, 2.        ])
```

subtract(*self*, *other*, **args*, ***kwargs*)

Perform element-wise 'subtract' operation with two inputs.

Parameters

- **self** (band_hic_matrix) – First input matrix.
- **other** (band_hic_matrix *or array-like*) – Second input for the operation.
- ***args** (*tuple*) – Additional positional arguments for numpy.subtract.
- ****kwargs** (*dict*) – Keyword arguments for numpy.subtract.

Returns
Result of element-wise 'subtract' operation.

Return type
band_hic_matrix

See also

numpy.subtract

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> other = mat.copy()
>>> result = mat.subtract(other)
```

sum(*axis: int | str | None = None*) → Number | ndarray

Compute the sum of the values in the matrix or along a given axis.

Parameters
axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the sum: - None: sum over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns
Sum(s) along the specified axis.

Return type
scalar or ndarray

Raises
ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.sum()  # sum of all elements
24
>>> mat.sum(axis='row')
array([ 1, 10, 13])
```

tan(*self*, **args*, ***kwargs*)

Perform element-wise 'tan' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.tan.
- ****kwargs** (*dict*) – Keyword arguments for numpy.tan.

Returns
Result of element-wise 'tan' operation.

Return type
band_hic_matrix

See also

numpy.tan

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.tan()
```

tanh(*self*, **args*, ***kwargs*)

Perform element-wise 'tanh' operation.

Parameters

- **self** (band_hic_matrix) – Input matrix.
- ***args** (*tuple*) – Additional positional arguments for numpy.tanh.
- ****kwargs** (*dict*) – Keyword arguments for numpy.tanh.

Returns
Result of element-wise 'tanh' operation.

Return type
band_hic_matrix

See also

numpy.tanh

Examples

```
>>> from bandhic import band_hic_matrix
>>> mat = band_hic_matrix(np.eye(3), diag_num=2, dtype=int)
>>> result = mat.tanh()
```

tocoo(*drop_zeros: bool = True*) → coo_array

Convert the matrix to COO format.

Parameters
drop_zeros (*bool, optional*) – If True, zero entries will be dropped from the COO format. Default is True.

Returns
The matrix in scipy COO sparse format.

Return type
coo_array

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> coo = mat.tocoo()
>>> coo.shape
(3, 3)
```

tocsr() → csr_array

Convert the matrix to CSR format.

Returns
The matrix in scipy CSR sparse format.

Return type
csr_array

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((3,3), diag_num=2)
>>> csr = mat.tocsr()
>>> csr.shape
(3, 3)
```

todense() → ndarray | MaskedArray

Convert the band matrix to a dense format.

Returns
The dense (square) matrix. Masked if mask is set.

Return type
ndarray or MaskedArray

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(3), diag_num=2)
>>> dense = mat.todense()
>>> dense.shape
(3, 3)
```

unmask(*indices: Tuple[ndarray, ndarray] | None = None*) → None

Remove mask entries for specified indices or clear all.

Parameters

indices (*tuple of array-like or None*) – Tuple (row_idx, col_idx) to unmask, or None to clear all.

Raises

Warning – If no mask exists when trying to remove.

Notes

If *indices* is None, this will clear all masks (both entry-level and row/column), equivalent to *clear_mask()*.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.eye(3), diag_num=2)
>>> mat.unmask(( [0],[0] ))
>>> mat.unmask()
```

var(*axis: int | str | None = None*) → Number | ndarray

Compute the variance of the values in the matrix or along a given axis.

Parameters

axis (*None, int, or {'row','col','diag'}, optional*) – Axis along which to compute the variance: - None: variance over all stored values (and default for missing). - 0 or 'row': per-row reduction. - 1 or 'col': per-column reduction. - 'diag': per-diagonal reduction. Default is None.

Returns

Variance(s) along the specified axis.

Return type

scalar or ndarray

Raises

ValueError – If axis is not one of the supported values.

Examples

```
>>> import bandhic as bh
>>> mat = bh.band_hic_matrix(np.arange(9).reshape(3,3), diag_num=2)
>>> mat.var()  # variance of all elements
7.555555555555555
>>> mat.var(axis='row')
array([ 0.22222222,  2.88888889, 10.88888889])
```

4.3 Utility Functions

bandhic.cooler_chr(file_path, chrom, diag_num)	Read Hi-C data from a .cool or .mcool file and return a band_hic_matrix.
bandhic.ones(shape[, diag_num, dtype])	Create a band_hic_matrix object filled with ones.
bandhic.zeros(shape[, diag_num, dtype])	Create a band_hic_matrix object filled with zeros.
bandhic.eye(shape[, diag_num, dtype])	Create a band_hic_matrix object filled as an identity matrix.
bandhic.full(shape, fill_value[, diag_num, ...])	Create a band_hic_matrix object filled with a specified value.
bandhic.ones_like(other[, dtype])	Create a band_hic_matrix object matching another matrix, filled with ones.
bandhic.zeros_like(other[, dtype])	Create a band_hic_matrix object matching another matrix, filled with zeros.
bandhic.eye_like(other[, dtype])	Create a band_hic_matrix object matching another matrix, filled as an identity matrix.
bandhic.full_like(other, fill_value[, dtype])	Create a band_hic_matrix object matching another matrix, filled with a specified value.

4.3.1 bandhic.cooler_chr

bandhic.**cooler_chr**(*file_path: str*, *chrom: str*, *diag_num: int*, *resolution: int | None = None*, *balance: bool = True*) → *band_hic_matrix*

Read Hi-C data from a .cool or .mcool file and return a band_hic_matrix.

Parameters

- **file_path** (*str*) – Path to the .cool, .mcool or .scool file.
- **chrom** (*str*) – Chromosome name.
- **diag_num** (*int*) – Number of diagonals to consider.
- **resolution** (*int, optional*) – Resolution of the Hi-C data.
- **balance** (*bool, optional*) – If True, use balanced data. Default is False. This parameter is specific to cooler files.

Returns

A band_hic_matrix object containing the Hi-C data.

Return type

band_hic_matrix

Raises

ValueError – If the cooler file is invalid or parameters are incorrect.

Examples

```
>>> import bandhic as bh
>>> mat = bh.cooler_chr('/Users/wwb/Documents/workspace/BandHiC-Master/data/yeast.
↪10kb.cool', 'chrI', resolution=10000, diag_num=10)
>>> isinstance(mat, band_hic_matrix)
True
```

See also

None

URL
https://cooler.readthedocs.io/en/latest/index.html

4.3.2 bandhic.ones

bandhic.**ones**(*shape: ~typing.Tuple[int, int]*, *diag_num: int = 1*, *dtype: type = <class 'numpy.float64'>*) → *band_hic_matrix*

Create a band_hic_matrix object filled with ones.

Parameters

- **shape** (*tuple of int*) – Matrix shape as (bins, bins).
- **diag_num** (*int, optional*) – Number of diagonals to consider. Default is 1.
- **dtype** (*data-type, optional*) – The data type of the matrix. Default is np.float64.

Returns
A band_hic_matrix object with all entries filled with ones.

Return type
band_hic_matrix

Examples

```
>>> import bandhic as bh
>>> mat = bh.ones((5, 5), diag_num=3)
>>> print(mat)
band_hic_matrix(shape=(5, 5), diag_num=3, dtype=<class 'numpy.float64'>)
```

4.3.3 bandhic.zeros

bandhic.**zeros**(*shape: ~typing.Tuple[int, int]*, *diag_num: int = 1*, *dtype: type = <class 'numpy.float64'>*) → *band_hic_matrix*

Create a band_hic_matrix object filled with zeros.

Parameters

- **shape** (*tuple of int*) – Matrix shape as (bins, bins).
- **diag_num** (*int, optional*) – Number of diagonals to consider. Default is 1.
- **dtype** (*data-type, optional*) – The data type of the matrix. Default is np.float64.

Returns
A band_hic_matrix object with all entries filled with zeros.

Return type
band_hic_matrix

Examples

```
>>> import bandhic as bh
>>> mat = bh.zeros((5, 5), diag_num=3)
>>> print(mat)
band_hic_matrix(shape=(5, 5), diag_num=3, dtype=<class 'numpy.float64'>)
```

4.3.4 bandhic.eye

bandhic.**eye**(*shape: ~typing.Tuple[int, int]*, *diag_num: int = 1*, *dtype: type = <class 'numpy.float64'>*) → *band_hic_matrix*

Create a band_hic_matrix object filled as an identity matrix.

Parameters

- **shape** (*tuple of int*) – Matrix shape as (bins, bins).
- **diag_num** (*int, optional*) – Number of diagonals to consider. Default is 1.
- **dtype** (*data-type, optional*) – The data type of the matrix. Default is np.float64.

Returns

A band_hic_matrix object with ones on the main diagonal and zeros elsewhere.

Return type

band_hic_matrix

Examples

```
>>> mat = eye((5, 5), diag_num=3)
>>> print(mat)
band_hic_matrix(shape=(5, 5), diag_num=3, dtype=<class 'numpy.float64'>)
```

4.3.5 bandhic.full

bandhic.**full**(*shape: ~typing.Tuple[int, int]*, *fill_value: int | float*, *diag_num: int = 1*, *dtype: type = <class 'numpy.float64'>*) → *band_hic_matrix*

Create a band_hic_matrix object filled with a specified value.

Parameters

- **shape** (*tuple of int*) – Matrix shape as (bins, bins).
- **fill_value** (*scalar*) – Value to fill the matrix with.
- **diag_num** (*int, optional*) – Number of diagonals to consider. Default is 1.
- **dtype** (*data-type, optional*) – The data type of the matrix. Default is np.float64.

Returns

A band_hic_matrix object with all entries filled with *fill_value*.

Return type

band_hic_matrix

Examples

```
>>> import bandhic as bh
>>> mat = bh.full((5, 5), fill_value=7, diag_num=3)
>>> print(mat)
band_hic_matrix(shape=(5, 5), diag_num=3, dtype=<class 'numpy.float64'>)
```

4.3.6 bandhic.ones_like

bandhic.**ones_like**(*other:* band_hic_matrix, *dtype=None*) → *band_hic_matrix*

Create a band_hic_matrix object matching another matrix, filled with ones.

Parameters
other (band_hic_matrix) – Reference matrix.

Returns
A band_hic_matrix object matching *other*, filled with ones.

Return type
band_hic_matrix

Examples

```
>>> mat_ref = zeros((4, 4), diag_num=2)
>>> mat = ones_like(mat_ref)
>>> isinstance(mat, band_hic_matrix)
True
```

4.3.7 bandhic.zeros_like

bandhic.**zeros_like**(*other:* band_hic_matrix, *dtype=None*) → *band_hic_matrix*

Create a band_hic_matrix object matching another matrix, filled with zeros.

Parameters
other (band_hic_matrix) – Reference matrix.

Returns
A band_hic_matrix object matching *other*, filled with zeros.

Return type
band_hic_matrix

Examples

```
>>> mat_ref = zeros((4, 4), diag_num=2)
>>> mat = zeros_like(mat_ref)
>>> isinstance(mat, band_hic_matrix)
True
```

4.3.8 bandhic.eye_like

bandhic.**eye_like**(*other:* band_hic_matrix, *dtype=None*) → *band_hic_matrix*

Create a band_hic_matrix object matching another matrix, filled as an identity matrix.

Parameters
other (band_hic_matrix) – Reference matrix.

Returns
A band_hic_matrix object matching *other*, filled as an identity matrix.

Return type
band_hic_matrix

Examples

```
>>> mat_ref = zeros((4, 4), diag_num=2)
>>> mat = eye_like(mat_ref)
>>> isinstance(mat, band_hic_matrix)
True
```

4.3.9 bandhic.full_like

bandhic.**full_like**(*other:* band_hic_matrix, *fill_value: int | float*, *dtype=None*) → *band_hic_matrix*

Create a band_hic_matrix object matching another matrix, filled with a specified value.

Parameters

- **other** (band_hic_matrix) – Reference matrix.
- **fill_value** (*scalar*) – Value to fill the matrix.

Returns
A band_hic_matrix object matching *other*, filled with *fill_value*.

Return type
band_hic_matrix

Examples

```
>>> mat_ref = zeros((4, 4), diag_num=2)
>>> mat = full_like(mat_ref, fill_value=9)
>>> isinstance(mat, band_hic_matrix)
True
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

Z