serpentine Documentation

Release 0.1.0

Vittore Scolari, Lyam Baudry

Mar 18, 2019

Contents:

1 Tutorial **1**
1.1 Serpentine binning tutorial 1

2 Demo **3**
2.1 Serpentine binning 3

3 Reference API **9**
3.1 serpentine 9

4 Indices and tables **17**

Python Module Index **19**

CHAPTER 1

Tutorial

1.1 Serpentine binning tutorial

This tutorial aims at demonstrating use cases and for improving Hi-C contact maps with distribution-aware binning and documenting readers with the implementation. For a detailed step-by-step analysis, see the *serpentine notebook*.

1.1.1 Dependencies

Python 3 with the following libraries:

- numpy
- matplotlib

1.1.2 Testing

Run the following:

```
serpentine --test --threshold 30 --min-threshold 3 --size 100
```

This randomly generates two 100x100 matrices of pixels between 0 and 10, binning both such that serpentines are not below 3 on average value in each of them and not below 30 in total value. It then plots both matrices as well as their log-ratio, binned and unbinned. Tweak the parameters to see how the matrices evolve.

1.1.3 Binning prepared Hi-C datasets

Run the following:

```
wget https://github.com/koszullab/serpentine/blob/master/demos/A.csv
wget https://github.com/koszullab/serpentine/blob/master/demos/B.csv
serpentine --inputs A.csv B.csv
```

This performs the same binning on prepared datasets from *Escherichia coli*. Currently supported formats for Hi-C datasets generated with DADE. They are essentially triangular, dense matrix files with the first row and the first column being used to indicate genomic position.

1.1.4 Using the library

You can directly use the library's functions if you are already manipulating numpy contact maps in your analysis. Open a Python 3 console, and run the following:

```
import numpy as np
from serpentine import serpentine_binning
from matplotlib import pyplot as plt

matrix3 = np.loadtxt("https://github.com/koszullab/serpentine/blob/master/demos/A.csv
↪", dtype=np.float64)
matrix4 = np.loadtxt("https://github.com/koszullab/serpentine/blob/master/demos/B.csv
↪", dtype=np.float64)
_, _, binned_matrix3, binned_matrix4, log_ratio, _ = serpentine_binning(matrix3,␣
↪matrix4)

plt.imshow(log_ratio, cmap='seismic')
plt.show()
```

This is useful if your contact maps come from other sources and aren't in a DADE format.

CHAPTER 2

Demo

2.1 Serpentine binning

This tutorial aims at demonstrating use cases and for improving Hi-C contact maps with distribution-aware binning, helping readers reproduce the steps in our papers and documenting readers with the implementation.

2.1.1 Loading the library

You can directly use the library's functions, if you are already manipulating numpy contact maps in your analysis. First, you need to import the library with the following:

[28]:
```
%matplotlib notebook
```

[29]:
```
import numpy as np
from matplotlib import pyplot as plt
import serpentine as sp
```

After, you need to load your datasets with numpy, we provide a couple of demo datasets in the form of tables, corresponding to Yeast chromosome 7 in two different mutants, in the repository:

[30]:
```
# Load Yeast data
A = np.loadtxt('../../demos/A.csv')
B = np.loadtxt('../../demos/B.csv')
```

At this point you are working with raw Hi-C data, serpentine provides convenient functions to visualize your material:

The first dataset:

[31]:
```
fig = plt.figure()
sp.mshow(A)
```

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

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

[31]:
```
<matplotlib.image.AxesImage at 0x7fea6aad2320>
```

The second dataset:

[32]:
```
fig = plt.figure()
sp.mshow(B)
```

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

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

[32]:
```
<matplotlib.image.AxesImage at 0x7fea6ac87cc0>
```

2.1.2 Filtering of the data

The raw data needs to be filtered in order to clean the unmappable rows and columns, this kind of artifacts shows up in the distribution of reads per bin as outliers:

[33]:
```
plt.figure()
norm = np.log10(np.sum(A + B, axis=0)[np.sum(A + B, axis=0) > 0])
norm = norm[np.isnan(norm) == False]
norm = norm[np.isinf(np.abs(norm)) == False]
plt.hist(norm, bins=50)
plt.axvline(x=np.median(norm), color='g')
plt.axvline(x=np.median(norm) - 3 * 1.4826 * sp.mad(norm), color='r')
plt.axvline(x=np.median(norm) + 3 * 1.4826 * sp.mad(norm), color='r')
```

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

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

[33]:
```
<matplotlib.lines.Line2D at 0x7fea6a852c50>
```

with the serpentine library this is easily done achievable using the two included functions

[34]:
```
flt = sp.outstanding_filter(A) + sp.outstanding_filter(B)
flt = flt == False
A = sp.fltmatr(A, flt)
B = sp.fltmatr(B, flt)
```

resulting in:

[35]:
```
fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1); sp.mshow(A, subplot=ax1)
ax2 = fig.add_subplot(1, 2, 2); sp.mshow(B, subplot=ax2)
```

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

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

[35]:
```
<matplotlib.image.AxesImage at 0x7fea6a796780>
```

at this point, additional processing can be done before proceeding, such as iterative normalizations, removing speckles and other such operations.

2.1.3 Finding the binning threshold and the de-trending constant

The coverage of the data will impact how much binning is needed. On top of that, when comparing matrices with different coverages, one needs to find the so-called trending constant that need to be subtracted from the result. In order to do this, our library provides a tool in the form of an mean-difference (MD) plot. This graph suggests that the data has a characteristic noise-to-signal ratio at large coverages that becomes much larger at lower coverages due to sampling noise.

The function MDbefore finds the optimal trending and threshold values, the graph higlights the median and the median absolute deviation (MAD) as red and green lines. Both are plotted as a function of the mean contact number:

[36]:
```
# Find the de-trending and threshold
plt.figure()
trend, threshold = sp.MDbefore(A, B, ylim=[-4, 4])
print(trend, threshold)
```

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

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

```
0.5109957290996449 49.999999999999986
```

2.1.4 Serpenting binning the data

Finally you can use the function to bin the data. The function takes two parameters: a threshold that constrains the coverage of the bin in at least one matrix, and the minthreshold that constrain it in both. The function uses multiple processors and can be configured by the optional parameters:

[37]:
```
sA, sB, sK = sp.serpentin_binning(A, B, threshold, threshold / 5)
```

```
Starting 10 binning processes in batches of 4...
0        Total serpentines: 40804 (100.0 %)
0        Total serpentines: 40804 (100.0 %)
0        Total serpentines: 40804 (100.0 %)
0        Total serpentines: 40804 (100.0 %)
1        Total serpentines: 28227 (69.1770414665229 %)
1        Total serpentines: 28188 (69.08146260170571 %)
1        Total serpentines: 28216 (69.1500833251642 %)
1        Total serpentines: 28315 (69.39270659739242 %)
2        Total serpentines: 12232 (29.977453190863642 %)
2        Total serpentines: 12248 (30.016665032839917 %)
2        Total serpentines: 12181 (29.85246544456426 %)
2        Total serpentines: 12288 (30.11469463778061 %)
3        Total serpentines: 7527 (18.446720909714735 %)
3        Total serpentines: 7475 (18.319282423291835 %)
3        Total serpentines: 7522 (18.434467209097146 %)
4        Total serpentines: 6530 (16.003333006567985 %)
4        Total serpentines: 6534 (16.013135967062052 %)
4        Total serpentines: 6533 (16.010685226938534 %)
3        Total serpentines: 7527 (18.446720909714735 %)
5        Total serpentines: 6459 (15.829330457798255 %)
5        Over: 2018-10-31 13:57:24.042602
5        Total serpentines: 6454 (15.817076757180669 %)
5        Total serpentines: 6450 (15.8072737966866 %)
6        Total serpentines: 6452 (15.812175276933633 %)
6        Over: 2018-10-31 13:57:24.099372
4        Total serpentines: 6522 (15.983727085579845 %)
6        Total serpentines: 6450 (15.8072737966866 %)
```

(continues on next page)

(continued from previous page)

```
6       Over: 2018-10-31 13:57:24.135747
5       Total serpentines: 6447 (15.799921576316047 %)
6       Total serpentines: 6447 (15.799921576316047 %)
6       Over: 2018-10-31 13:57:24.195925
0       Total serpentines: 40804 (100.0 %)
0       Total serpentines: 40804 (100.0 %)
0       Total serpentines: 40804 (100.0 %)
0       Total serpentines: 40804 (100.0 %)
1       Total serpentines: 28225 (69.17213998627585 %)
1       Total serpentines: 28226 (69.17459072639937 %)
1       Total serpentines: 28246 (69.22360552886971 %)
1       Total serpentines: 28194 (69.09616704244682 %)
2       Total serpentines: 12231 (29.975002450740124 %)
3       Total serpentines: 7515 (18.417312028232526 %)
2       Total serpentines: 12236 (29.98725615135771 %)
2       Total serpentines: 12225 (29.96029800999902 %)
4       Total serpentines: 6509 (15.95186746397412 %)
5       Total serpentines: 6426 (15.748456033722183 %)
5       Over: 2018-10-31 13:57:24.859513
3       Total serpentines: 7526 (18.444270169591217 %)
2       Total serpentines: 12245 (30.009312812469364 %)
3       Total serpentines: 7542 (18.483482011567492 %)
4       Total serpentines: 6525 (15.991079305950397 %)
4       Total serpentines: 6522 (15.983727085579845 %)
5       Total serpentines: 6453 (15.81462601705715 %)
5       Total serpentines: 6444 (15.792569355945496 %)
3       Total serpentines: 7550 (18.503087932555633 %)
6       Total serpentines: 6453 (15.81462601705715 %)
6       Over: 2018-10-31 13:57:25.030012
6       Total serpentines: 6444 (15.792569355945496 %)
6       Over: 2018-10-31 13:57:25.035744
4       Total serpentines: 6525 (15.991079305950397 %)
5       Total serpentines: 6448 (15.802372316439564 %)
6       Total serpentines: 6447 (15.799921576316047 %)
6       Over: 2018-10-31 13:57:25.130998
0       Total serpentines: 40804 (100.0 %)
0       Total serpentines: 40804 (100.0 %)
1       Total serpentines: 28313 (69.38780511714538 %)
1       Total serpentines: 28130 (68.93931967454171 %)
2       Total serpentines: 12176 (29.84021174394667 %)
3       Total serpentines: 7532 (18.45897461033232 %)
4       Total serpentines: 6509 (15.95186746397412 %)
2       Total serpentines: 12253 (30.028918733457505 %)
5       Total serpentines: 6452 (15.812175276933633 %)
6       Total serpentines: 6450 (15.8072737966866 %)
6       Over: 2018-10-31 13:57:25.767884
3       Total serpentines: 7567 (18.544750514655426 %)
4       Total serpentines: 6581 (16.128320752867367 %)
5       Total serpentines: 6493 (15.912655621997843 %)
6       Total serpentines: 6490 (15.905303401627291 %)
6       Over: 2018-10-31 13:57:25.921796
```

2.1.5 Checking the results

The quality of the binning can be verified with an MD plot. This time, use the MDafter function: if the process was successful you would expect the effect of sampling to be reduced by binning, and an almost constant signal-to-noise

value at all coverage values, similar to the one at large coverage:

[38]:
```
plt.figure()
sp.MDafter(sA, sB, sK, ylim=[-4, 4])
```

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

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

[38]:
```
(0.5107324222723643, 49.999999999999986)
```

Matrices have been rebinned, and the characteristic sampling noise present at small coverages has now been smoothed, while the crisp signal at large coverages that conveys the precious biological variations is preserved:

[39]:
```
fig = plt.figure();
ax1 = fig.add_subplot(1, 2, 1)
sp.mshow(sA, subplot=ax1)
ax2 = fig.add_subplot(1, 2, 2)
sp.mshow(sB, subplot=ax2)
```

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

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

[39]:
```
<matplotlib.image.AxesImage at 0x7fea6adc91d0>
```

2.1.6 Checking the differential analysis

Similarly, we improved the differential analysis, before the binning, we could have obtained this kind of results:

Before binning:

[40]:
```
plt.figure()
np.warnings.filterwarnings('ignore')
D = np.log2(B/A)
sp.dshow(D, trend)
```

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

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

Now,

After binning:

[41]:
```
plt.figure()
sp.dshow(sK, trend)
```

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

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

[27]:
```
from scipy.ndimage import gaussian_filter
sA, sB, sK = sp.serpentin_binning(A, B, threshold, threshold / 5)
```

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

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

[ ]:

[ ]:

CHAPTER 3

Reference API

3.1 serpentine

3.1.1 serpentine package

Submodules

serpentine.serpentine module

Serpentine binning

An implementation of the so-called 'serpentine binning' procedure described in Scolari et al.

Command line:

```
Usage:
    serpentine.py [<matrixA>] [<matrixB>] [--threshold=auto] [--verbose]
                  [--min-threshold=auto] [--trend=high] [--triangular]
                  [--limit=3] [--demo] [--demo-size=500]

Arguments:
    matrixA                          The first input matrix, in plain text
                                     CSV format. Optional in demo mode.
    matrixB                          The second input matrix, in plain text
                                     CSV format. Optional in demo mode or
                                     single binning mode.

Options:
    -h, --help                       Display this help message.
    --version                        Display the program's current version.
    -t auto, --threshold auto        Threshold value to trigger binning.
                                     [default: auto]
    -m auto, --min-threshold auto    Minimum value to force trigger binning
                                     in either matrix. [default: auto]
```

(continues on next page)

(continued from previous page)

```
    --trend high                    Trend to subtract to the differential
                                    matrix, possible values are "mean":
                                    equal amount of positive and negative
                                    differences, and "high": normalize
                                    at the regions with higher coverage.
                                    [default: high]
    --triangular                    Treat the matrix as triangular,
                                    useful when plotting matrices adjacent
                                    to the diagonal. [default: False]
    --limit 3                       Set the z-axis limit on the
                                    plot of the differential matrix.
                                    [default: 3]
    --demo                          Run a demo on randomly generated
                                    matrices. [default: False]
    --demo-size 500                 Size of the test matrix for the demo.
                                    [default: 500]
    -v, --verbose                   Show verbose output. [default: False]
```

serpentine.serpentine.**MDafter**(*XA*, *XB*, *XD*, *s=10*, *xlim=None*, *ylim=None*, *triangular=False*, *show=True*)

MD plot after binning

The MD plot is the main metric provided by the serpentin binning package, it visualized the variability in function of the mean coverage of a couple of matrices to be compared. This version is optimized to be run after the serpentin binning. The return values of this function should be generally ignored.

Parameters

- **XB** (*XA,*) – The input matrices
- **XD** (*array_like*) – The differential matrix obtained by serpentin binnning
- **s** (*int, optional*) – How many point use for the trend lines, depends on statistics and range
- **xlim** (*float, optional*) – Limits for the x-axis
- **ylim** (*float, optional*) – Limits for the y-axis
- **triangular** (*bool, optional*) – Set triangular if you are interested in rebin only half of the matrix (for instance in the case of matrices which are already triangular, default is false)
- **show** (*bool, optional*) – Set it to false if you are not interested in the graph but only in the return values of this function.

Returns **trend, threshold** – Normally something which should not bother you

Return type float

serpentine.serpentine.**MDbefore**(*XA*, *XB*, *s=10*, *xlim=None*, *ylim=None*, *triangular=False*, *show=True*)

MD plot before binning

The MD plot is the main metric provided by the serpentin binning package, it visualized the variability in function of the mean coverage of a couple of matrices to be compared. This version is optimized to be run before the serpentin binning. The return values of this funciton will be hints on the values to use to normalize the differential matrix and as a threshold for the serpentin binning algorithm.

Parameters

- **XB** (*XA,*) – The input matrices

- **s** (*int, optional*) – How many point use for the trend lines, depends on statistics and range
- **xlim** (*float, optional*) – Limits for the x-axis
- **ylim** (*float, optional*) – Limits for the y-axis
- **triangular** (*bool, optional*) – Set triangular if you are interested in rebin only half of the matrix (for instance in the case of matrices which are already triangular, default is false)
- **show** (*bool, optional*) – Set it to false if you are not interested in the graph but only in the return values of this function.

Returns

- **trend** (*float*) – This is the extrapolated trend of the two matrices, it is measured by the ratio of the most covered part of the two matrices, use it to set the zero to the output of the differential analysis if you think this works better than the np.mean() function.
- **threshold** (*float*) – If the statistics permits the automatical extimation of the threshold, this will be a good parameter to use as input to the serpentin binning algorithm

serpentine.serpentine.**all_barycenters**(*M_serp*, *weights=None*)

Compute all serpentine barycenters

Extract all serpentines from a serpentinized matrix, then compute all serpentine barycenters, optionally weigthed by the non-serpentinized matrix.

Parameters

- **M_serp** (*numpy.ndarray*) – The serpentinized matrix
- **weights** (*numpy.ndarray or None, optional*) – The non-serpentinized (original) matrix acting as weights. Default is None.

serpentine.serpentine.**barycenter**(*serp*, *weights=None*)

Compute weighted serpentine barycenter

Compute the (optionally weighted) barycenter of a serpentine, where the weights would be equal to the values of the map itself.

Parameters

- **serp** (*iterable*) – An iterable of serpentine coordinates
- **weights** (*numpy.ndarray or None, optional*) – If None, the barycenter is unweighted. Otherwise, if it is a contact map, the barycenter is weighted by the values of the map at the serpentine's coordinates. Default is None.

Returns **bary** – The barycenter coordinates

Return type tuple

serpentine.serpentine.**dshow**(*dif*, *trend*, *limit=3*, *triangular=False*, *colorbar=True*, *cmap=None*, *ax=<module 'matplotlib.pyplot' from '/home/docs/checkouts/readthedocs.org/user_builds/serpentine/envs/latest/lib/python3.7/site-packages/matplotlib-3.0.3-py3.7-linux-x86_64.egg/matplotlib/pyplot.py'>*)

Show differential matrix

A boilerplate around the imshow matplotlib function to show the differential matrix

Parameters

- **dif** (*array_like*) – The differential matrix

- **trend** (*float*) – The value of the zero, please use either the output of MDbefore function or the value of md.mean(dif)
- **limit** (*float, optional*) – The colorscale limit of the log-ratio, setting it to 2 or 3 seems like a sensible choice. Defaults to 3
- **triangular** (*bool, optional*) – Set triangular if you are interested in rebin only half of the matrix (for instance in the case of matrices which are already triangular, default is false)
- **cmap** (*str, optional*) – Color map of the plotted matrix. Should be ideally diverging, default is sesismic.
- **ax** (*optional*) – Set axis, defaults to matplotlib library

Returns

Return type The plot

serpentine.serpentine.**extract_serpentines**(*M*)

Extract serpentine structure

Isolate serpentines based on shared pixel values in a contact map.

Parameters **M** (*numpy.ndarray*) – The (serpentinized) input contact map

serpentine.serpentine.**fltmatr**(*X: numpy.ndarray*, *flt: numpy.ndarray*) → numpy.ndarray

Filter a 2D matrix in both dimensions according to an boolean vector.

Parameters

- **X** (*array_like*) – The input matrix
- **flr** (*array_like*) – The boolean filter

Returns The filtered matrix

Return type array_like

Example

```
>>> import numpy as np
>>> M = np.ones((5, 5))
>>> M[2:4, 2:4] = 2
>>> print(M)
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  2.  2.  1.]
 [ 1.  1.  2.  2.  1.]
 [ 1.  1.  1.  1.  1.]]
>>> flt = M.sum(axis=1) > 5
>>> X = fltmatr(M, flt)
>>> print(X)
[[ 2.  2.]
 [ 2.  2.]]
```

serpentine.serpentine.**fromupdiag**(*filename*)

Load a DADE matrix into a numpy array

serpentine.serpentine.**mad**(*data: numpy.ndarray*, *axis: Optional[int] = None*) → numpy.float64

Median absolute deviation

Calculates the median absolute deviation of data

Parameters

- **data** (*array_like*) – The dataset
- **axis** (*int, optional*) – The axis over which perform the numpy.median function

Returns The median absolute deviation

Return type float

serpentine.serpentine.**mshow**(*XX*, *subplot=<module 'matplotlib.pyplot' from '/home/docs/checkouts/readthedocs.org/user_builds/serpentine/envs/latest/lib/python3.7/site-packages/matplotlib-3.0.3-py3.7-linux-x86_64.egg/matplotlib/pyplot.py'>*, *colorbar=True*, *triangular=False*)

Boilerplate around the imshow function to show a matrix.

serpentine.serpentine.**outstanding_filter**(*X: numpy.ndarray*) → numpy.ndarray

Generate filtering index that removes outstanding values (three standard MADs above or below the median).

Parameters **X** (*array_like*) – The dataset

Returns The boolean filter

Return type array_like

Example

```
>>> import numpy as np
>>> X = np.arange(25).reshape((5,5))
>>> X += X.T
>>> X[2,2] += 10000
>>> print(X)
[[    0     6    12    18    24]
 [    6    12    18    24    30]
 [   12    18 10024    30    36]
 [   18    24    30    36    42]
 [   24    30    36    42    48]]
>>> O = outstanding_filter(X)
>>> print(O)
[False False  True False False]
```

serpentine.serpentine.**serpentin_binning**(*A: numpy.ndarray*, *B: numpy.ndarray*, *threshold: float = 50.0*, *minthreshold: float = 10.0*, *iterations: float = 10.0*, *triangular: bool = False*, *verbose: bool = True*, *parallel: int = 4*, *sizes: bool = False*) → Tuple

Perform the serpentin binning

The function will perform the algorithm to serpentin bin two matrices, iterations can be done in series or in parallel, convinient for multi-processor machines.

Parameters

- **B** (*A,*) – The matrices to be compared.
- **threshold** (*float, optional*) – The threshold of rebinning for the highest coverage matrix. Default is set by the DEFAULT_THRESHOLD parameter, which is 50 if unchanged.

- **minthreshold** (*float, optional*) – The threshold for both matrices. Default is set by the DEFAULT_MIN_THRESHOLD parameter, which is 10 if unchanged.
- **iterations** (*int, optional*) – The number of iterations requested, more iterations will consume more time, but also will result in better and smoother results. Default is set by the DEFAULT_ITERATIONS parameter, which is 10 if unchanged.
- **triangular** (*bool, optional*) – Set triangular if you are interested in rebinning only half of the matrix (for instance in the case of matrices which are already triangular, default is False).
- **verbose** (*bool, optional*) – Whether to print additional output during the computation. Default is False.
- **parallel** (*int, optional*) – Set it to the number of your processor if you want to attain maximum speeds. Default is 4.
- **sizes** (*bool, optional*) – Whether to keep track of the serpentine size distribution, in which case it will be returned as a Counter. Default is False.

Returns

- **sA, sB** (*array_like*) – The rebinned matrices
- **sK** (*array_like*) – The log-ratio matrix, expressend in base 2. Attention, the matrix needs to be normalized by subtracting an appropriate value for the zero (MDbefore or numpy.mean functions are there to help you in this task).
- **serp_size_distribution** (*collections.Counter, optional*) – A counter keeping track of the serpentine size distribution. Only returned if the supplied 'sizes' parameter is True.

serpentine.serpentine.**serpentin_iteration**(*A: numpy.ndarray*, *B: numpy.ndarray*, *threshold: float = 50.0*, *minthreshold: float = 10.0*, *triangular: bool = False*, *verbose: bool = True*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Perform a single iteration of serpentin binning

Each serpentin binning is generally executed in multiple iterations in order to smooth the random variability in the bin aggregation. This funciton performs a single iteration.

Parameters

- **B** (*A,*) – The matrices to be compared.
- **threshold** (*float, optional*) – The threshold of rebinning for the highest coverage matrix.
- **minthreshold** (*float, optional*) – The threshold for both matrices
- **triangular** (*bool, optional*) – Set triangular if you are interested in rebin only half of the matrix (for instance in the case of matrices which are already triangular, default is false)
- **verbose** (*bool, optional*) – Set it false if you are annoyed by the printed output.

Returns

- **Amod, Bmod** (*array_like*) – The rebinned matrices
- **D** (*array_like*) – The log-ratio matrix, expressend in base 2. Attention, the matrix need to be normalized by subtractiong an appropriate value for the zero (MDbefore or numpy.mean functions are there to help you in this task).

serpentine.version module

Module contents

Serpentine binning for Hi-C contact maps

Provides: -An implementation of the serpentine binning procedure described in Scolari et al., usable on single or multiple contact maps -A set of functions for quickly visualizing the effects of serpentine binning in terms of contact distribution (median absolute deviation, etc.)

serpentine.**gaussian_blurring**(*input*, ***, *sigma=1*, *order=0*, *output=None*, *mode='reflect'*, *cval=0.0*, *truncate=4.0*)

Multidimensional Gaussian filter.

Parameters

- **input** (*array_like*) – The input array.
- **sigma** (*scalar or sequence of scalars*) – Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **order** (*int or sequence of ints, optional*) – The order of the filter along each axis is given as a sequence of integers, or as a single number. An order of 0 corresponds to convolution with a Gaussian kernel. A positive order corresponds to convolution with that derivative of a Gaussian.
- **output** (*array or dtype, optional*) – The array in which to place the output, or the dtype of the returned array. By default an array of the same dtype as input will be created.
- **mode** (*str or sequence, optional*) – The *mode* parameter determines how the input array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the input array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The input is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The input is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The input is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The input is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The input is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0.
- **truncate** (*float*) – Truncate the filter at this many standard deviations. Default is 4.0.

Returns **gaussian_filter** – Returned array of same shape as *input*.

Return type ndarray

Notes

The multidimensional filter is implemented as a sequence of one-dimensional convolution filters. The intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited precision, the results may be imprecise because intermediate results may be stored with insufficient precision.

Examples

```
>>> from scipy.ndimage import gaussian_filter
>>> a = np.arange(50, step=2).reshape((5,5))
>>> a
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28],
       [30, 32, 34, 36, 38],
       [40, 42, 44, 46, 48]])
>>> gaussian_filter(a, sigma=1)
array([[ 4,  6,  8,  9, 11],
       [10, 12, 14, 15, 17],
       [20, 22, 24, 25, 27],
       [29, 31, 33, 34, 36],
       [35, 37, 39, 40, 42]])
```

```
>>> from scipy import misc
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.gray()  # show the filtered result in grayscale
>>> ax1 = fig.add_subplot(121)  # left side
>>> ax2 = fig.add_subplot(122)  # right side
>>> ascent = misc.ascent()
>>> result = gaussian_filter(ascent, sigma=5)
>>> ax1.imshow(ascent)
>>> ax2.imshow(result)
>>> plt.show()
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

serpentine, 15
serpentine.serpentine, 9
serpentine.version, 15

Index

A

all_barycenters() (in module serpentine.serpentine), 11

B

barycenter() (in module serpentine.serpentine), 11

D

dshow() (in module serpentine.serpentine), 11

E

extract_serpentines() (in module serpentine.serpentine), 12

F

fltmatr() (in module serpentine.serpentine), 12
fromupdiag() (in module serpentine.serpentine), 12

G

gaussian_blurring() (in module serpentine), 15

M

mad() (in module serpentine.serpentine), 12
MDafter() (in module serpentine.serpentine), 10
MDbefore() (in module serpentine.serpentine), 10
mshow() (in module serpentine.serpentine), 13

O

outstanding_filter() (in module serpentine.serpentine), 13

S

serpentin_binning() (in module serpentine.serpentine), 13
serpentin_iteration() (in module serpentine.serpentine), 14
serpentine (module), 15
serpentine.serpentine (module), 9
serpentine.version (module), 15