blockify

Release 0.2.2

Arnav Moudgil

Sep 22, 2020

CONTENTS

Blockify is a command line program and Python library for genome segmentation and peak calling using Bayesian blocks.

CHAPTER

ONE

INTRODUCTION

blockify is a fast and optimal genomic peak caller for one-dimensional data (e.g. BED, qBED, CCF).

The package is built around the Bayesian blocks algorithm [SNJC13], which finds the optimal change points in time series data assuming a Poisson counting process. We also implement a dynamic pruning strategy which achieves linear runtime performance [KFE12]. An interactive notebook demonstrating Bayesian blocks can be found here.

While Bayesian blocks was originally developed in the astrophysics community for photon-counting experiments, we find that it has applications in genomics. In particular, we use it to analyze transposon calling cards experiments. Calling cards uses a transposase fused to a transcription factor (TF) to deposit transposons near TF binding sites. Bayesian blocks partitions the genome based on the local density of insertions, which in turn are used to identify peaks and candidate TF binding sites. We have also had success using this algorithm to perfom general-purpose genome segmentation.

Note: Recent papers using calling cards include [SLC+19] and [LSM20]. For examples of Bayesian blocks in practice, see [CMC+20] and [MWC+20].

---

blockify is best designed to process qBED files [MLH+20], although it will work with BED files.

To get started, please see our *Installation* guide and *Tutorial*.

1.1 References

CHAPTER

TWO

INSTALLATION

blockify runs on Python (>= 3.4) and is installable via `pip`

```
pip install blockify
```

2.1 Development

To actively develop blockify, clone from GitHub and switch to the development branch:

```
git clone https://github.com/arnavm/blockify.git
cd blockify
git checkout dev
```

Unit tests are available from the top-level directory:

```
python -m unittest tests.test_basic
```

Two batteries of tests are provided: `tests.test_basic` and `tests.test_advanced`. For routine development, the basic set of tests should be sufficient. The advanced suite takes much more time and fetches several large datasets. It is best used when making major changes to the code.

2.2 Disclaimer

Not to be confused with the similarly-named Spotify plugin.

CHAPTER

THREE

TUTORIAL

This tutorial illustrates basic blockify usage. We will analyze some previously published data [MWC+20]. Specifically, we will segment bulk SP1-*piggyBac* calling cards data and call SP1-directed peaks. We will also segment wild-type *piggyBac* calling cards data and call BRD4-directed peaks.

3.1 Getting Started

We need to download and decompress the processed data files, as well as a reference set of TTAA tetramers.

```
> wget -O HCT-116_PBase.ccf.gz "https://www.ncbi.nlm.nih.gov/geo/download/?
↪acc=GSM4471636&format=file&file=GSM4471636%5FHCT%2D116%5FPBase%2Eccf%2Etxt%2Egz"
> gunzip HCT-116_PBase.ccf.gz
> wget -O HCT-116_SP1-PBase.ccf.gz "https://www.ncbi.nlm.nih.gov/geo/download/?
↪acc=GSM4471637&format=file&file=GSM4471637%5FHCT%2D116%5FSP1%2DPBase%2Eccf%2Etxt
↪%2Egz"
> gunzip HCT-116_SP1-PBase.ccf.gz
> wget https://gitlab.com/arnavm/calling_cards/-/raw/master/Ref/TTAA/hg38_TTAA.bed.xz
> xz -d hg38_TTAA.bed.xz
> ls
HCT-116_PBase.ccf
HCT-116_SP1-PBase.ccf
hg38_TTAA.bed
```

Here, HCT-116_SP1-PBase.ccf are the insertions from the SP1-directed experiment, HCT-116_PBase.ccf are the insertions from the wild-type transposase, and hg38_TTAA.bed are the set of potential *piggyBac* insertion sites.

3.2 Calling SP1 Peaks

We first segment the experiment file as this gives us a candidate set of regions with piecewise-constant densities. These contiguous, bookended intervals are known as blocks.

```
> blockify segment -i HCT-116_SP1-PBase.ccf -o HCT-116_SP1-PBase.blocks
> wc -l HCT-116_SP1-PBase.blocks
21375
> head -n 3 HCT-116_SP1-PBase.blocks
chr1 54672   758707
chr1 758707  906326
chr1 906326  906925
```

We now use this set of blocks along with the insertions themselves to call peaks. Since the blocks have piecewise-constant density, we model the number of insertions in each block as a Poisson process. For each block, we parameterize the background Poisson process using the control, undirected insertions, scaled for library size. A set of peaks might be called like this:

```
> blockify call -i HCT-116_SP1-PBase.ccf -r HCT-116_SP1-PBase.blocks -bg HCT-116_
↪PBase.ccf -a 0.05 --correction fdr_bh -d 250 -t -o HCT-116_SP1-PBase_peaks.bed
> wc -l HCT-116_SP1-PBase_peaks.bed
8356
```

Here, we specified the input file with `-i`, the input regions with `-r`, and the background data with `-bg`. We used Benjamini-Hochberg correction at a false-discovery rate of 5% (`-a 0.05 --correction fdr_bh`), merging significant blocks within 250 bp of each other (`-d 250`), and tightening the final peaks (`-t`) to improve peak resolution. For a full list of options, see *Command: call*.

3.3 Calling BRD4 Peaks

To identify BRD4-bound peaks from undirected insertions, we follow a similar set of commands for calling TF-directed peaks. There are two key differences: first, we use hg38_TTAA.bed as the background file, as our null model assumes insertions would be uniformly distributed across the genome; and second, we set the pseudocount to 0 (`-c 0`). When calling TF-directed peaks, both the input and background sets of insertions are random variables. Thus, in any given block, it is possible that there are zero insertions in the background file. To account for undersampling, we added a pseudocount (default: 1). This is not necessary when calling BRD4 peaks because there will always be a TTAA in each block, as *piggyBac* virtually always inserts into this motif. Retaining a pseudocount, while not technically wrong, will decrease sensitivity.

```
> blockify call -i HCT-116_PBase.ccf -r HCT-116_PBase.blocks -bg hg38_TTAA.bed -c 0 -
↪p 1e-30 -d 12500 -o HCT-116_PBase_peaks.bed
> wc -l HCT-116_PBase_peaks.bed
1935
```

Here, we've optimized peak calling to detect BRD4-bound super-enhancers. We've set a very strict threshold, using an unadjusted p-value cutoff of 1e-30 (`-p 1e-30`) and merging significant blocks within 12,500 bp (`-d 12500`).

CHAPTER

FOUR

TIPS AND TRICKS

Here are some helpful tips and tricks to get the most out of blockify.

4.1 Resolution

Transposition data can be sparse, particularly if the transposase is constricted to specific motifs (e.g. *piggyBac*). Sparse data lead to broader peaks, which can be harder to interpret. Here are some strategies to increase resolution:

- Omit or decrease the `-d/--distance` flag to minimize merging of significant blocks
- Specify `-t/--tight`, which will pull in peak boundaries so they overlap qBED entries
- For the sharpest intervals, use the `-s/--summit` flag to return each peak's maximum
- Finally, increasing the value of `--p0` (default: 0.05) can lead to more peaks being called, at the risk of returning more false positives.

4.2 Miscellaneous

- Although the *Tutorial* demonstrated first generating a list of blocks for input into `blockify call`, this step is not strictly necessary. If a regions file is not supplied, blockify will generate one behind the scenes using the default settings in `blockify segment`. However, this can result in considerable memory usage. Pre-computing the blocks file is one way to minimize memory consumption and improve performance.
- Similarly, the regions over which to run `blockify call` need not be Bayesian blocks. The program can operate on any set of intervals provided in BED format. This flexibility can be useful if there are a set of features that are biologically meaningful to your analysis. For example, this could be a file of promoter regions or accessible loci where a TF might be bound.
- Peaks are output in BED6 format with a generic annotation, like `peak_1743`. The program does not re-calculate *post hoc* p-values on peaks. If you want to further calculate the significance or normalized density of these peaks, simply re-run `blockify call` with the `--intermediate` flag set and supply the peaks file to `-r/--regions`. Then inspect the intermediate file for these details. Picking up with the BRD4 example from the *Tutorial*:

```
> blockify call -i HCT-116_PBase.ccf -r HCT-116_PBase_peaks.bed -bg hg38_TTAA.bed -c␣
↪0 -p 1e-30 -d 12500 --intermediate HCT-116_PBase_peaks_annotated.csv > /dev/null
> head -n 2 HCT-116_PBase_peaks_annotated.csv
,chrom,start,end,name,score,strand,Input,Background,Normed_bg,Net_density,pValue,
↪negLog10pValue,rejected
0,chr1,7298597,7304456,peak_0,1,.,130.0,32,2.533130940448789,0.021755738020063357,3.
↪74243334103237e-169,168.42684592642368,True
```

CHAPTER

FIVE

BLOCKIFY

Genomic peak caller for one-dimensional data

```
usage: blockify [-h] [-v] {segment,normalize,call,downsample} ...
```

5.1 Positional Arguments

command Possible choices: segment, normalize, call, downsample

Subcommands

5.2 Named Arguments

-v, --version show program's version number and exit

CHAPTER

SIX

COMMAND: SEGMENT

Segment a BED/qBED file using Bayesian blocks

```
usage: blockify segment [-h] -i INPUT [--prior PRIOR | --p0 P0]
                        [--method {OP,PELT}]
                        output
```

6.1 Named Arguments

-i, --input	Input file
output	Output file (BED format); default: stdout
--prior	Explicit prior on the number of blocks (not recommended for general use)
--p0	Empirical prior based on a specified false-positive rate; must be between 0 and 1 (default: 0.05)
--method	Possible choices: OP, PELT Segment using the optimal partitioning (OP) or pruned exact linear time (PELT) algorithm (default: "PELT")

CHAPTER

SEVEN

COMMAND: CALL

Call peaks in a qBED file

```
usage: blockify call [-h] -i INPUT [--prior PRIOR | --p0 P0]
                     [--method {OP,PELT}] [-r REGIONS] -bg BACKGROUND
                     [--intermediate INTERMEDIATE]
                     (-a ALPHA | -p PVALUECUTOFF) [--correction CORRECTION]
                     [-d DISTANCE] [--min MIN] [--max MAX] [-t | -s]
                     [-c PSEUDOCOUNT] [--measure {enrichment,depletion}]
                     output
```

7.1 Named Arguments

-i, --input	Input file
output	Output file (BED format); default: stdout
--prior	Explicit prior on the number of blocks (not recommended for general use)
--p0	Empirical prior based on a specified false-positive rate; must be between 0 and 1 (default: 0.05)
--method	Possible choices: OP, PELT Segment using the optimal partitioning (OP) or pruned exact linear time (PELT) algorithm (default: "PELT")
-r, --regions	Regions over which to normalize event counts; should be supplied as a BED file. If not provided, the input file will be segmented using Bayesian blocks.
-bg, --background	Background qBED file
--intermediate	Intermediate file to write verbose output (CSV format)
-a, --alpha	Alpha for multiple hypothesis correction (must be between 0 and 1)
-p, --pValueCutoff	p-value cutoff (NOTE: This is a straight cutoff and will not take into account multiple hypothesis correction!)
--correction	If alpha provided, need to specificity method of multiple hypothesis correction. See statsmodels.stats.multitest for a complete list of choices (default: "bonferroni")
-d, --distance	Merge features closer than this distance (bp)
--min	Report peaks larger than this cutoff (bp)
--max	Report peaks smaller than this cutoff (bp)

-t, --tight	Shrink peak boundaries to overlap data points
-s, --summit	Return peak summits
-c, --pseudocount	Pseudocount for background regions (default: 1)
--measure	Possible choices: enrichment, depletion Perform a one-tailed test for either enrichment or depletion relative to the background file (default: “enrichment”)

CHAPTER

EIGHT

COMMAND: NORMALIZE

Calculate normalized rates of events in a qBED file

```
usage: blockify normalize [-h] -i INPUT [--prior PRIOR | --p0 P0]
                          [--method {OP,PELT}] [-r REGIONS] [-o OUTPUT]
                          [-k LIBRARYFACTOR] [-l LENGTHFACTOR]
```

8.1 Named Arguments

-i, --input	Input file
--prior	Explicit prior on the number of blocks (not recommended for general use)
--p0	Empirical prior based on a specified false-positive rate; must be between 0 and 1 (default: 0.05)
--method	Possible choices: OP, PELT Segment using the optimal partitioning (OP) or pruned exact linear time (PELT) algorithm (default: "PELT")
-r, --regions	Regions over which to normalize event counts; should be supplied as a BED file. If not provided, the input file will be segmented using Bayesian blocks.
-o, --output	Output file (bedGraph format); default: stdout
-k, --libraryFactor	Normalization factor for library size (default: 1000000)
-l, --lengthFactor	Normalization factor for the length of regions; used to calculate scaled rates of events per interval (default: None)

CHAPTER

NINE

COMMAND: DOWNSAMPLE

Downsample a qBED file in proportion to the value column

```
usage: blockify downsample [-h] -i INPUT -n NUMBER [-s SEED] [--naive]
                           [-o OUTPUT]
```

9.1 Named Arguments

-i, --input	Input file
-n, --number	Number of entries to downsample to (cannot exceed length of input file)
-s, --seed	Random seed
--naive	Sample every row with equal likelihood
-o, --output	Output file (BED/qBED format); default: stdout

CHAPTER

TEN

API: SEGMENTATION

class blockify.segmentation.**SegmentationRecord**
A class to store a single Bayesian block genomic segmentation.

finalize()
Store post hoc summary statistics of the segmentation.

blockify.segmentation.**blocksToDF**(*chrom*, *ranges*)
Convert a set of contiguous Bayesian blocks to pandas DataFrame format.

Parameters

- **chrom** (*str*) – String specifying the chromsome
- **ranges** (*array*) – Array whose entries specify the coordinates of block boundaries

Returns **output**

Return type pandas DataFrame

blockify.segmentation.**segment**(*input_file*, *method*, *p0=None*, *prior=None*)
Core segmentation method.

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **method** (*str*) – String specifying whether to use OP or PELT for the segmentation
- **p0** (*float, optional*) – Float used to parameterize the prior on the total number of blocks; must be in the interval [0, 1]. Default: 0.05
- **prior** (*float, optional*) – Explicit value for the total number of priors (specifying this is not recommended)

Returns **segmentation** – A SegmentationRecord from segmenting the provided data

Return type *SegmentationRecord*

blockify.segmentation.**segment_from_command_line**(*args*)
Wrapper function for the command line function blockify segment

Parameters **args** (argparse.Namespace object) – Input from command line

Returns **segmentation** – A SegmentationRecord from segmenting the command line data

Return type *SegmentationRecord*

blockify.segmentation.**validateSegmentationArguments**(*input_file*, *p0*, *prior*)
Validates parameters passed via the command line.

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **p0** (*float*) –
- **prior** (*float*) –

Returns None

Return type None

CHAPTER

ELEVEN

API: ANNOTATION (PEAK CALLING)

blockify.annotation.**annotate**(*input_file*, *regions_bed*, *background_file*, *measure='enrichment'*, *intermediate=None*, *alpha=None*, *correction=None*, *p_value=None*, *distance=None*, *min_size=None*, *max_size=None*, *pseudocount=1*, *tight=False*, *summit=False*)

Core annotation and peak calling method.

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **regions_bed** (*BedTool object*) – BedTool object (instantiated from pybedtools) for regions over which we are annotation/calling peaks
- **background_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) used to parameterize the background model
- **measure** (*str*) – Either "enrichment" or "depletion" to indicate which direction of effect to test for
- **intermediate** (*bool*) – Whether or not to return intermediate calculations during peak calling
- **alpha** (*float or None*) – Multiple-hypothesis adjusted threshold for calling significance
- **correction** (*str or None*) – Multiple hypothesis correction to perform (see `statsmodels.stats.multitest` for valid values)
- **p_value** (*float or None*) – Straight p-value cutoff (unadjusted) for calling significance
- **distance** (*int or None*) – Merge significant features within specified distance cutoff
- **min_size** (*int or None*) – Minimum size cutoff for peaks
- **max_size** (*int or None*) – Maximum size cutoff for peaks
- **pseudocount** (*float*) – Pseudocount added to adjust background model
- **tight** (*bool*) – Whether to tighten the regions in `regions_bed`
- **summit** (*bool*) – Whether to return peak summits instead of full peaks

Returns

- **out_bed** (*BedTool object*) – Set of peaks in BED6 format
- **df** (`pandas` DataFrame or None) – If `intermediate` specified, DataFrame containing intermediate calculations during peak calling

blockify.annotation.**annotate_from_command_line**(*args*)
Wrapper function for the command line function `blockify call`

Parameters **args** (`argparse.Namespace` object) – Input from command line

Returns

- **out_bed** (*BedTool object*) – Set of peaks in BED6 format
- **df** (`pandas` DataFrame or None) – If `intermediate` specified, DataFrame containing intermediate calculations during peak calling

blockify.annotation.**getPeakSummits**(*df*, *metric='pValue'*)
From a list of peaks, get a set of peak summits

Parameters

- **df** (`pandas` DataFrame) – Set of peaks from `annotate` as a DataFrame
- **metric** (*str*) – Metric to use when filtering for summits. One of "pValue" or "density"

Returns **summits** – Set of peak summits as a DataFrame

Return type `pandas` DataFrame

blockify.annotation.**parcelConsecutiveBlocks**(*df*)
Concatenates consecutive blocks into a DataFrame. If there are multiple non-contiguous sets of consecutive blocks, creates one DataFrame per set.

Parameters **df** (`pandas` DataFrame) – Input set of blocks as a DataFrame

Returns **outlist** – List of DataFrames, each of which is a set of consecutive blocks

Return type list of `pandas` DataFrames

blockify.annotation.**sizeFilter**(*bed*, *min_size*, *max_size*)
Filter peaks by size.

Parameters

- **bed** (*BedTool object*) – Input data file
- **min_size** (*int*) – Lower bound for peak size
- **max_size** (*int*) – Upper bound for peak size

Returns **filtered_peaks** – Peaks after size selection

Return type BedTool object

blockify.annotation.**tighten**(*data*)
Tightens block boundaries in a BedTool file. This function modifies block boundaries so that they coincide with data points.

Parameters **data** (*BedTool object*) – Input file of block boundaries

Returns **refined** – BedTool of tightened blocks

Return type BedTool object

blockify.annotation.**validateAnnotationArguments**(*input_file*, *regions_bed*, *background_file*, *measure*, *alpha*, *correction*, *p_value*, *distance*, *min_size*, *max_size*, *pseudocount*)
Validates parameters passed via the command line.

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **regions_bed** (*BedTool object*) – BedTool object (instantiated from pybedtools) for regions over which we are annotation/calling peaks
- **background_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) used to parameterize the background model
- **measure** (*str*) – Either "enrichment" or "depletion" to indicate which direction of effect to test for
- **alpha** (*float or None*) – Multiple-hypothesis adjusted threshold for calling significance
- **correction** (*str or None*) – Multiple hypothesis correction to perform (see `statsmodels.stats.multitest` for valid values)
- **p_value** (*float or None*) – Straight p-value cutoff (unadjusted) for calling significance
- **distance** (*int or None*) – Merge significant features within specified distance cutoff
- **min_size** (*int or None*) – Minimum size cutoff for peaks
- **max_size** (*int or None*) – Maximum size cutoff for peaks
- **pseudocount** (*float*) – Pseudocount added to adjust background model

Returns None

Return type None

CHAPTER

TWELVE

API: NORMALIZATION

blockify.normalization.**normalize**(*input_file*, *regions_bed*, *libraryFactor*, *lengthFactor*)

Core normalization method

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **regions_bed** (*BedTool object*) – BedTool object (instantiated from pybedtools) for regions over which we are normalizing input_file
- **libraryFactor** (*float*) – Scalar to normalize by input_file's library size.
- **lengthFactor** (*float or None*) – Scalar to normalize by each block's length. If None, no length normalization is performed.

Returns **bedgraph** – A BedTool object in bedGraph format, using the intervals supplied in regions_bed

Return type BedTool

blockify.normalization.**normalize_from_command_line**(*args*)

Wrapper function for the command line function `blockify normalize`

Parameters **args** (`argparse.Namespace` object) – Input from command line

Returns **bedgraph** – Normalized command line data in bedGraph format

Return type BedTool

blockify.normalization.**validateNormalizationArguments**(*input_file*, *regions_bed*, *libraryFactor*, *lengthFactor*)

Validates parameters passed via the command line.

Parameters

- **input_file** (*BedTool object*) – BedTool object (instantiated from pybedtools) for input data
- **regions_bed** (*BedTool object*) – BedTool object (instantiated from pybedtools) for regions over which we are normalizing input_file
- **libraryFactor** (*float*) – Scalar to normalize by input_file's library size.
- **lengthFactor** (*float or None*) – Scalar to normalize by each block's length. If None, no length normalization is performed.

Returns **None**

Return type None

CHAPTER

THIRTEEN

API: DOWNSAMPLING

blockify.downsampling.**downsample**(*input_file*, *n*, *seed=None*, *naive=False*)

Core downsampling method

Parameters

- **input_file** (pandas DataFrame) – Input data (e.g. BED, qBED, CCF) as a pandas DataFrame
- **n** (*int*) – Number of entries to sample
- **seed** (*int*) – Seed for random number generator
- **naive** (*bool*) – Choose whether to sample each entry with equal probability (True) or weighted by the value in the fourth column (if supplied)

Returns **downsampled_file** – Input file after downsampling

Return type BedTool object

blockify.downsampling.**downsample_from_command_line**(*args*)

Wrapper function for the command line function blockify downsample

Parameters **args** (argparse.Namespace object) – Input from command line

Returns **downsampled_file** – Downsampled command line data

Return type BedTool

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[CMC+20] Alexander J. Cammack, Arnav Moudgil, Jiayang Chen, Michael J. Vasek, Mark Shabsovich, Katherine McCullough, Allen Yen, Tomas Lagunas, Susan E. Maloney, June He, Xuhua Chen, Misha Hooda, Michael N. Wilkinson, Timothy M. Miller, Robi D. Mitra, and Joseph D. Dougherty. A viral toolkit for recording transcription factor–DNA interactions in live mouse tissues. *Proceedings of the National Academy of Sciences*, pages 201918241, April 2020. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1918241117, doi:10.1073/pnas.1918241117.

[KFE12] R Killick, P Fearnhead, and I A Eckley. Optimal Detection of Changepoints With a Linear Computational Cost. *Journal of the American Statistical Association*, 107(500):1590–1598, October 2012. URL: https://www.tandfonline.com/doi/full/10.1080/01621459.2012.737745, doi:10.1080/01621459.2012.737745.

[LSM20] Jiayue Liu, Christian A Shively, and Robi D Mitra. Quantitative analysis of transcription factor binding and expression using calling cards reporter arrays. *Nucleic Acids Research*, 48(9):e50–e50, May 2020. URL: https://academic.oup.com/nar/article/48/9/e50/5781211, doi:10.1093/nar/gkaa141.

[MLH+20] Arnav Moudgil, Daofeng Li, Silas Hsu, Deepak Purushotham, Ting Wang, and Robi David Mitra. The qBED track: a novel genome browser visualization for point processes. preprint, bioRxiv, April 2020. URL: http://biorxiv.org/lookup/doi/10.1101/2020.04.27.060061, doi:10.1101/2020.04.27.060061.

[MWC+20] Arnav Moudgil, Michael N. Wilkinson, Xuhua Chen, June He, Alexander J. Cammack, Michael J. Vasek, Tomás Lagunas, Zongtai Qi, Matthew A. Lalli, Chuner Guo, Samantha A. Morris, Joseph D. Dougherty, and Robi D. Mitra. Self-Reporting Transposons Enable Simultaneous Readout of Gene Expression and Transcription Factor Binding in Single Cells. *Cell*, pages S009286742030814X, July 2020. shortDOI:d4wx. URL: https://linkinghub.elsevier.com/retrieve/pii/S009286742030814X, doi:10.1016/j.cell.2020.06.037.

[SNJC13] Jeffrey D Scargle, Jay P Norris, Brad Jackson, and James Chiang. STUDIES IN ASTRONOMICAL TIME SERIES ANALYSIS. VI. BAYESIAN BLOCK REPRESENTATIONS. *The Astrophysical Journal*, 764(2):167, February 2013. URL: http://stacks.iop.org/0004-637X/764/i=2/a=167?key=crossref.0539dc6f37f29e250567031865ebbe9a, doi:10.1088/0004-637X/764/2/167.

[SLC+19] Christian A. Shively, Jiayue Liu, Xuhua Chen, Kaiser Loell, and Robi D. Mitra. Homotypic cooperativity and collective binding are determinants of bHLH specificity and function. *Proceedings of the National Academy of Sciences*, 116(32):16143–16152, August 2019. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1818015116, doi:10.1073/pnas.1818015116.

PYTHON MODULE INDEX

b

INDEX