CHESSBOARD

Release 0.1

David Wang

Oct 18, 2022

CONTENTS

CHESSBOARD is a python implementation of the algorithm described in *Wang et al. 2022*. It also comes with an API to facilitate preprocessing of raw data and exploration of algorithm output.

Check out the *Usage* section for further information, including how to install the project.

To see an example workflow, check out the Jupyter Notebook at *CHESSBOARD Basic Usage Tutorial*.

To visualize data, check out the GAMBIT tool

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Usage

1.1.1 Installation

Use the package manager [pip](https://pip.pypa.io/en/stable/) to install CHESSBOARD and the API.

```
$ pip install git+https://bitbucket.org/biociphers/chessboard
```

1.1.2 Usage

CHESSBOARD is a command line tool that can be run as follows using the provided arguments. Note that the input is a CHESSBOARD `.cb` file which can be generated using our API. The API allows the user to import raw data for processing prior to running CHESSBOARD and explore the algorithm output for downstream analsysis.

```
usage: chessboard [-h] [-A CONC] [-L REG] [-I ITER] [-B BURNIN] [-T THIN]
                  [-S BLOCKSIZE] [-M minsize] [-K kinit] [-V verbose] [--version]
                  input out_dir

positional arguments:
  input                 Path to the chessboard input file.
  out_dir               Path to the output directory; will be created if it
                        does not already exist.

optional arguments:
  -h, --help            show this help message and exit
  -A CONC, --conc CONC  Concentration parameter. Larger values increase the
                        number of learned clusters. (default: 1e-100)
  -L REG, --reg REG     Regularization parameter. (default: 50)
  -I ITER, --iter ITER  Number of iterations to run the Gibbs sampler.
                        (default: 1000)
  -B BURNIN, --burnin BURNIN
                        Number of burn-in iterations. (default: 200)
  -T THIN, --thin THIN  Number of samples to discard between iterations to
                        reduce autocorrelation. (default: 2)
  -S BLOCKSIZE, --blocksize BLOCKSIZE
                        Maximum block size for blocked Gibbs sampling
                        procedure. (default: 10)
  -M MINSIZE, --minsize MINSIZE
```

(continues on next page)

(continued from previous page)

```
            Minimum tile size. (default: 20)

  -K KINIT, --kinit KINIT
            Number of clusters for k-means initialization. (default: 5)

  -V, --verbose
            Verbose mode. (default: False)
  --version             show program's version number and exit
```

1.2 API

Import CHESSBOARD as:

```
import chessboard.api as cb
```

1.2.1 Read and Write: `io`

Functions for reading and writing data. Users need to read input data into a CHESSBOARD *io.Data* object to perform all downstream operations, data manipulation and analysis. CHESSBOARD currently accept 2 input data types.

1. MAJIQ builder `.majiq` files. These files contain junction spanning read counts from MAJIQ [Vaquero-Garcia16].
2. Equivalent data processed through another junction mapping software in the form of `.tsv` files.

io.Data(reads, samples, lsvs, psi)	CHESSBOARD object for storing and manipulating data.
io.createFromMajiq(dir, outfilename[, ...])	Process and save MAJIQ generated data.
io.loadData(file)	Load MAJIQ generated data.
io.loadTSV(file1, file2)	Load data from `.tsv` files.
io.saveData(data, file)	Save the current state of a CHESSBOARD object.
io.loadCBObject(file)	Load saved CHESSBOARD object.

chessboard.api.io.Data

class chessboard.api.io.**Data**(*reads*, *samples*, *lsvs*, *psi*)

CHESSBOARD object for storing and manipulating data.

Stores splicing data and allows user to generate summary statistics and perform processing operations on the data.

Parameters

- **reads** (ndarray) – A 2 x lsv x sample matrix containing junction spanning read outs. On the first axis, index 0 represents the number of junction spanning reads and index 1 represents the number of other reads in the normalization unit (e.g. LSV, intron cluster, entire gene, etc).
- **samples** (ndarray) – Vector of sample names.
- **lsvs** (ndarray) – Vector of LSV IDs.
- **psi** (ndarray) – A lsv x sample matrix with PSI quantifications.

__init__(*reads*, *samples*, *lsvs*, *psi*)

Methods

__init__(reads, samples, lsvs, psi)	
addChessboardOutput(filename)	Load results of CHESSBOARD algorithm into object.
subsetLSVs(index)	Retain only LSVs at the specified indicies.
subsetSamples(index)	Retain only samples at the specified indicies.
writeChessboardInput(filename)	Save data for input to CHESSBOARD algorithm.
writeGambitFile(filename)	Save data for input to GAMBIT visualization tool.

Attributes

num_lsvs	Number of LSVs
num_samples	Number of samples

chessboard.api.io.createFromMajiq

chessboard.api.io.**createFromMajiq**(*dir*, *outfilename*, *quant=10*, *miss=0.8*)

Process and save MAJIQ generated data.

Extracts junction spanning read rates from MAJIQ [Vaquero-Garcia16] files and saves them in `.npz` format for downstream analysis. The `.npz` file is saved in *outfilename*.

Parameters

- **dir** (str) – Path to the `.majiq` files generated using the MAJIQ software.
- **outfilename** (str) – Path to the output `.npz` file.
- **quant** (int) – Minimum number of reads for junction to be quantifiable.
- **miss** (float) – Maximum fraction of samples for which a LSV is allowed to be missing before being removed.

chessboard.api.io.loadData

chessboard.api.io.**loadData**(*file*)

Load MAJIQ generated data.

Loads MAJIQ generated data processed by *chessboard.api.io.createFromMajiq()* in `.npz` format as a CHESSBOARD data object.

Parameters
file (str) – Path to the `.npz` file.

Returns
A CHESSBOARD object containg the input data.

Return type
chessboard.api.io.Data

chessboard.api.io.loadTSV

chessboard.api.io.**loadTSV**(*file1*, *file2*)

Load data from `.tsv` files.

Load junction spanning read counts produced by other software in `.tsv` format. Each column of the `.tsv` is a sample and each row is a splice junction. `file1` should contain the reads mapped to the junction and `file2` should contain all other reads in the normalization unit (e.g. LSV, intron cluster, entire gene etc). Missing values should agree in position between both files.

Parameters

- **file1** (str) – Path to `.tsv` file containing junction spanning read counts.
- **file2** (str) – Path to `.tsv` containing reads mapped to all other junctions in the splicing normalization unit.

Returns
A CHESSBOARD object containing the input data.

Return type
chessboard.api.io.Data

chessboard.api.io.saveData

chessboard.api.io.**saveData**(*data*, *file*)

Save the current state of a CHESSBOARD object.

Save the current state of a CHESSBOARD object using pickle which can be loaded using :func: chessboard.api.io.loadCBObject

Parameters

- **data** (*Data*) – CHESSBOARD object to be saved.
- **file** (str) – Path to output file.

chessboard.api.io.loadCBObject

chessboard.api.io.**loadCBObject**(*file*)

Load saved CHESSBOARD object.

Load a pickled CHESSBOARD object.

Parameters
file (str) – Path to saved object crated using :func: chessboard.api.io.saveData.

1.2.2 Data Preprocessing: `prefilter`

Functions for prefiltering informative LSVs as described in *Wang et al. 2022*.

prefilter.VarFilter(data, var_thresh)	Variance filter
prefilter.PBSFilter(data[, n_boot, alpha])	PBS KS Filter

chessboard.api.prefilter.VarFilter

chessboard.api.prefilter.**VarFilter**(*data*, *var_thresh*)

Variance filter

Filters the LSVs in a Chessboard object by variance up to some variance threshold.

Parameters

- **data** (*Data*) – Chessboard object.
- **var_thresh** (float) – Variance threshold. LSVs with variance across the samples above this value are retained.

chessboard.api.prefilter.PBSFilter

chessboard.api.prefilter.**PBSFilter**(*data*, *n_boot=500*, *alpha=0.05*)

PBS KS Filter

Filters LSVs using a parametric bootstrap Kolmogorov-Smirnov test.

Parameters

- **data** – Chessboard object.
- **n_boot** – Number of bootstrap samples.
- **alpha** – p-value cutoff for the statistical test.

1.2.3 Prior Estimation: priors

Functions for estimating priors.

priors.EstimateMissingPriors(data, background)	

chessboard.api.priors.EstimateMissingPriors

chessboard.api.priors.**EstimateMissingPriors**(*data*, *background*, *downsample=2000*)

Estimate missing value priors.

Empirically estimate missing value model priors for the signal and background groups. In both cases, the priors are estimated using Beta-Binomial regression.

$$\upsilon_j \sim BetaBinomial(n, \mu\Phi, (1\mu)\Phi) logit(\mu) = \beta_0 + \beta_1\chi_j$$

Here, υ_j is the missingness rate and χ_j is the median read depth for a given LSV j.

For the signal, υ_j and χ_j are derived from the data. For the background, these quantities are obtained from control data. For example, GTEX data for the equivalent tissue. We provide sample data for GTEX whole blood in "gtex_missingness.tsv". This file should have a column "miss" representing υ_j and a column "mdepth" representing χ_j.

Parameters

- **data** (*Data*) – A CHESSBOARD object containg the data on which you need to estimate priors.
- **background** (str) – Path to the TSV file for estimating background priors.
- **downsample** (int) – Number of samples in the TSV to use. This is down sampling is to help reduce run time.

1.2.4 Posterior Summary: postsum

Functions for summarizing posterior samples.

postsum.generatePointEstimate(data, minprob)	Generate a clustering point estimate from MCMC samples.

chessboard.api.postsum.generatePointEstimate

chessboard.api.postsum.**generatePointEstimate**(*data*, *minprob*, *force_k=None*)

Generate a clustering point estimate from MCMC samples.

When a single tile configuration is of interest, the MCMC samples from CHESSBOARD need to be summarized into a point estimate. This method clusters the pairwise sample clustering probability matrix (i.e. the probability that any 2 samples are clustered together across MCMC samples) using heirarchical clustering where k is the median number clusters across sampled tile configurations to obtain sample assignments. LSV assignments are obtained by averaging the marginal probabilities of each matrix entry being in the signal group across all samples in a cluster for the given LSV. Posteriors of the distributional parameters are updated based on sample and LSV assignments.

Parameters

- **data** (*Data*) – Chessboard object. Must contain CHESSBOARD output after using chessboard.api.io.Data.addChessboardOutput().
- **minprob** – The minimum probability of the average margin to be considered signal.

1.2.5 Downstream Analysis: analysis

Functions for summarizing posterior samples.

analysis.outputLSVLists(data, file)	Output Excel .xlsx file containing summary statistics
analysis.computeLikelihood(data, x, r)	Compute likelihood on held out test data.

chessboard.api.analysis.outputLSVLists

chessboard.api.analysis.**outputLSVLists**(*data*, *file*)

Output Excel .xlsx file containing summary statistics

Outputs .xlsx file with the following tabs: 1. For each discovered tile, a tab showing the samples and LSVs in the tile along with the marignal probability of sample and LSV assignment to the tile. 2. A tab containing all LSVs that did not appear in any cluster (consensus background). 3. A tab showing missing value signals. For each LSV, if the signal group had an excess of missing values, show the increased missingness rate and fisher p-value of missingness enrichment.

Parameters

- **index** – CHESSBOARD object containing the data. CHESSBOARD output must be added and summarized. See *chessboard.api.io()* and *chessboard.api.postsum()*
- **file** (str) – Path to output file.

chessboard.api.analysis.computeLikelihood

chessboard.api.analysis.**computeLikelihood**(*data*, *x*, *r*)

Compute likelihood on held out test data.

Once the CHESSBOARD model is trained on a dataset and distributional parameters are learned, one can make predictions on held out data. This function returns the likelihood of cluster assignments given a sample vector and tile assignment vector. The tile vector is a binary vector of the sample length as the LSVs in the sample vector which indicate whether each LSV is signal (1) or background (0).

Parameters

- **data** (*Data*) – Chessboard object. Must contain CHESSBOARD output after using chessboard.api.io.Data.addChessboardOutput().
- **x** (ndarray) – A vector containing the junction spanning read counts of a sample. The first axis is the junction read counts (index 0) and alternate junction read counts (index 1). The second axis is LSVs. The total number of LSVs must be sample as the data used to train CHESSBOARD.
- **r** (ndarray) – Binary vector that indicates whether each sample in the LSV is signal (1) or background (0).

Returns
Vector of log likelihoods where each entry represents assignment to one of the k clusters.

Return type
ndarray

1.2.6 Visualization: vis

Visualization tools.

vis.PlotECDF(data)	ECDF plot
vis.PlotPointEstimate(data[, save])	Heatmap cluster plot
vis.PlotPointEstimate(data[, save])	Heatmap cluster plot
vis.Heatmap(data[, row_labels, col_labels, ...])	Heatmap plot
vis.Histogram(data, i)	LSV Histogram

chessboard.api.vis.PlotECDF

chessboard.api.vis.**PlotECDF**(*data*)

ECDF plot

Plots a empirical CDF (ECDF) of the Ψ variances for LSVs across samples. Use this to help determine reasonable variance cutoffs for filtering.

Parameters

data (*Data*) – Chessboard object.

chessboard.api.vis.PlotPointEstimate

chessboard.api.vis.**PlotPointEstimate**(*data*, *save=None*)

Heatmap cluster plot

Plots a heatmap representation of the data and clustering. Note that Ψ values are shown in the heatmap and white spaces are missing values. CHESSBOARD output should be added to the object.

Parameters

- **data** (*Data*) – Chessboard object. Must contain CHESSBOARD output after using `chessboard.api.io.Data.addChessboardOutput()`.
- **save** (`Optional[str]`) – Path to plot output save. If `None`, the plot is not saved.

chessboard.api.vis.Heatmap

chessboard.api.vis.**Heatmap**(*data*, *row_labels=None*, *col_labels=None*, *savefig=None*, *xlab='Samples'*, *ylab='LSV'*)

Heatmap plot

Plots a heatmap representation of the data without clustering. Note that Ψ values are shown in the heatmap and white spaces are missing values. Custom assignments can be added.

Parameters

- **data** (*Data*) – Chessboard object.
- **row_labels** – Vector of custom LSV assignments. This should be a binary matrix with n_cluster columns and n_lsv rows. A (1) indicates the the LSV is signal in the corresponding cluster.
- **col_labels** – Vector of integer labels for samples
- **savefig** – Path to plot output save. If `None`, the plot is not saved.
- **xlab** – x-axis label
- **ylab** – y-axis label

chessboard.api.vis.Histogram

chessboard.api.vis.**Histogram**(*data*, *i*)

LSV Histogram

Shows a histogram of the indicated LSV with the signal and background separated using a stacked bar plot and the posterior distribution curves drawn.

Parameters

- **data** (*Data*) – Chessboard object. Must contain CHESSBOARD output after using `chessboard.api.io.Data.addChessboardOutput()`.
- **i** (int) – Index of the LSV.

1.3 CHESSBOARD Basic Usage Tutorial

This tutorial shows how to use the CHESSBOARD API to process and analyze data with CHESSBOARD (Wang et al. 2021).

1.3.1 Load package

[1]:
```
from chessboard.api import io
from chessboard.api import prefilter
from chessboard.api import priors
from chessboard.api import postsum
from chessboard.api import analysis
from chessboard.api import vis
```

1.3.2 Load data

CHESSBOARD supports loading data in 2 ways. 1. Using MAJIQ: We can extract junction spanning read counts from BAM files using MAJIQ. Simply run MAJIQ on your BAM files to generate .majiq files. Then run the following on your .majiq file directory to generate an input npz file. The advantage of using this approach is that MAJIQ using bootstrapped junction spanning read rates which are more accurate than raw reads.

```
io.createFromMajiq("/majiq/files/dir/")
```

This .npz file can be loaded as a CHESSBOARD object using

[2]:
```
sim_data = io.loadData("data/sim_data.npz")
```

2. Using a pair of TSVs: If you wish to obtain junction spanning read counts using another software, simply use CHESSBOARD's TSV input format. The format of the TSVs is as follows:
   - First column is splicing unit names
   - Columns are sample names
   - Entries of first matrix represent representative junction/isoform read counts.
   - Entries of second matrix represent alternative junction/isoform read counts.

The user will be responsible for parsing their input from another program into this format.

[3]:
```
sim_data = io.loadTSV("data/sim_tsv_format_1.tsv","data/sim_tsv_format_2.tsv")
```

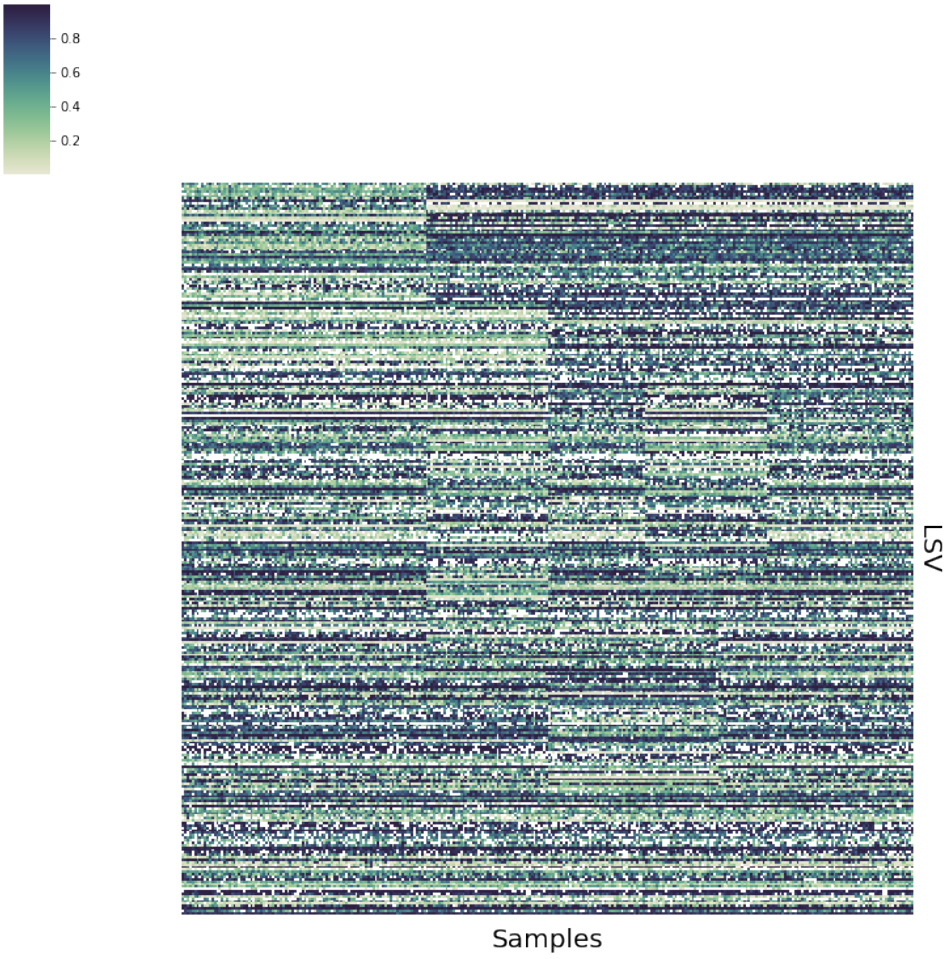
1.3.3 Exploratory Analysis

Check your data dimnsions and visualize the pre-clustered matrix.

[4]:
```
print("Dataset has", sim_data.num_samples(), "samples and", sim_data.num_lsvs(), "lsvs.")
```

```
Dataset has 300 samples and 300 lsvs.
```

[5]:
```
vis.Heatmap(sim_data.psi,savefig="plots/sim_data.png")
```



nbsphinx-code-borderwhite

1.3.4 Estimating Missingness Priors

This step requires training data from control (i.e non cancer tissue) tissues to fit a beta-binomial regression to empirically estimate priors. We provide training data for whole blood. If your tissue of interest is different, please generate a file where the first column is the missingness rate, second column is median read depth and each row is an LSV (or any unit of splicing measurement of your choice).

[6]:
```
priors.EstimateMissingPriors(sim_data,"data/gtex_missingness.csv")
```

1.3.5 Running CHESSBOARD Algorithm

To run the CHESSBOARD algorithm, save the current object and then running the command line tool. The tool can be run using

```
python chessboard.py data/sim_data.cb data/sim_data/ -I 100 -B 50 -V
```

For testing, we recommend running for a few hundred iterations in verbose mode. See the CHESSBOARD documentation or additional flags.

After running the algorithm, add the output back to the object.

[7]:
```
sim_data.writeChessboardInput("data/sim_data")
```

[8]:
```
sim_data.addChessboardOutput("data/sim_data_out.cb")
```

1.3.6 Posterior Summary

Summarize the posterior samples into a point estimate for plotting and downstream analysis. We recommend using a posterior marginal cutoff of 0.7 confidence.

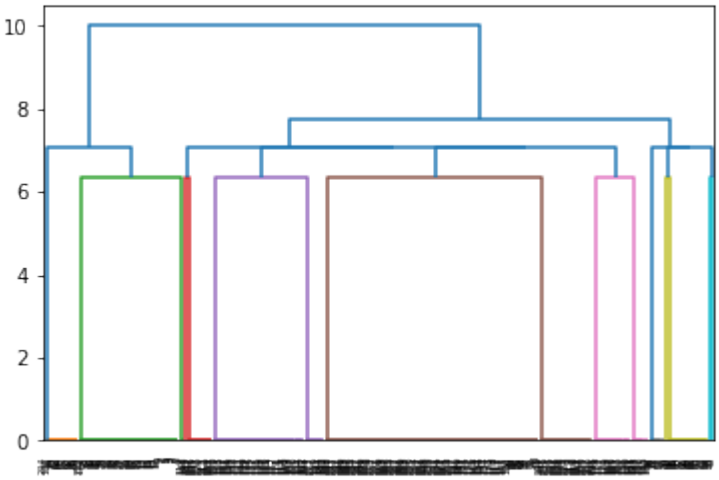
[9]:
```
postsum.generatePointEstimate(sim_data,0.7)
```

1.3.7 Visualization

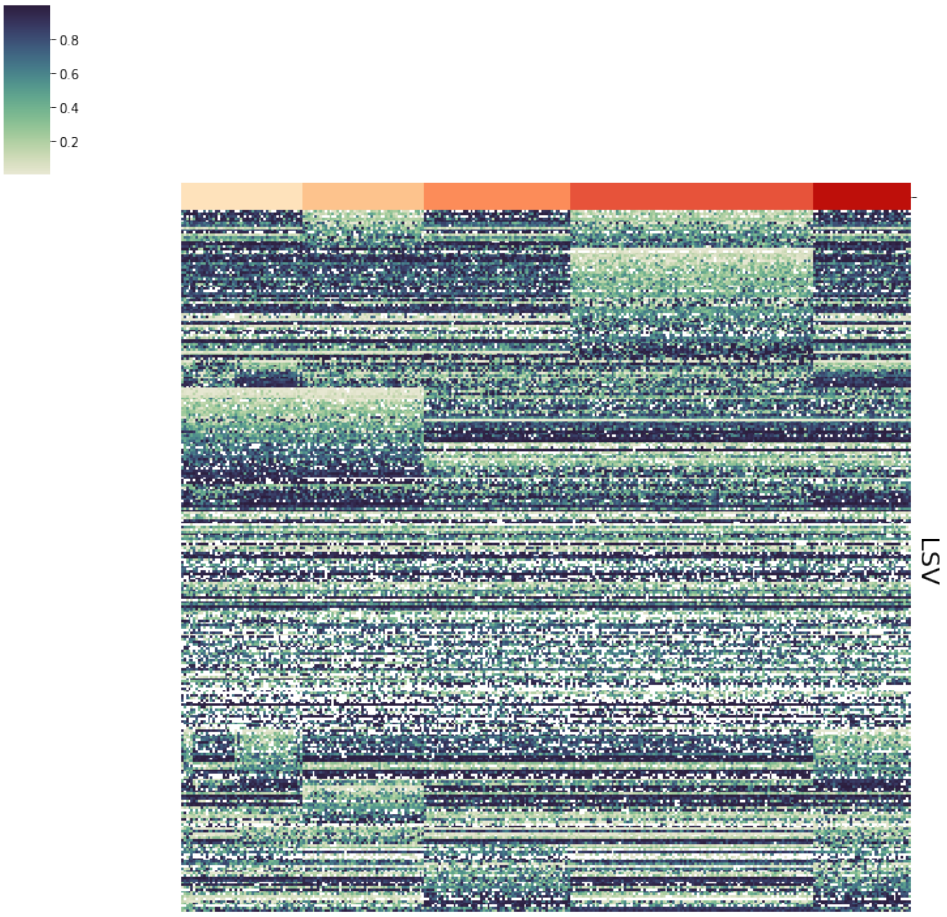
To visualize the result, we provide 2 methods: 1. GAMBIT: Simply write the object to GABMIT format and upload to our online interactive interface at: [URL HERE] 2. Python: the the function shown below and save plots.

[10]:
```
sim_data.writeGambitFile("data/sim_data")
```

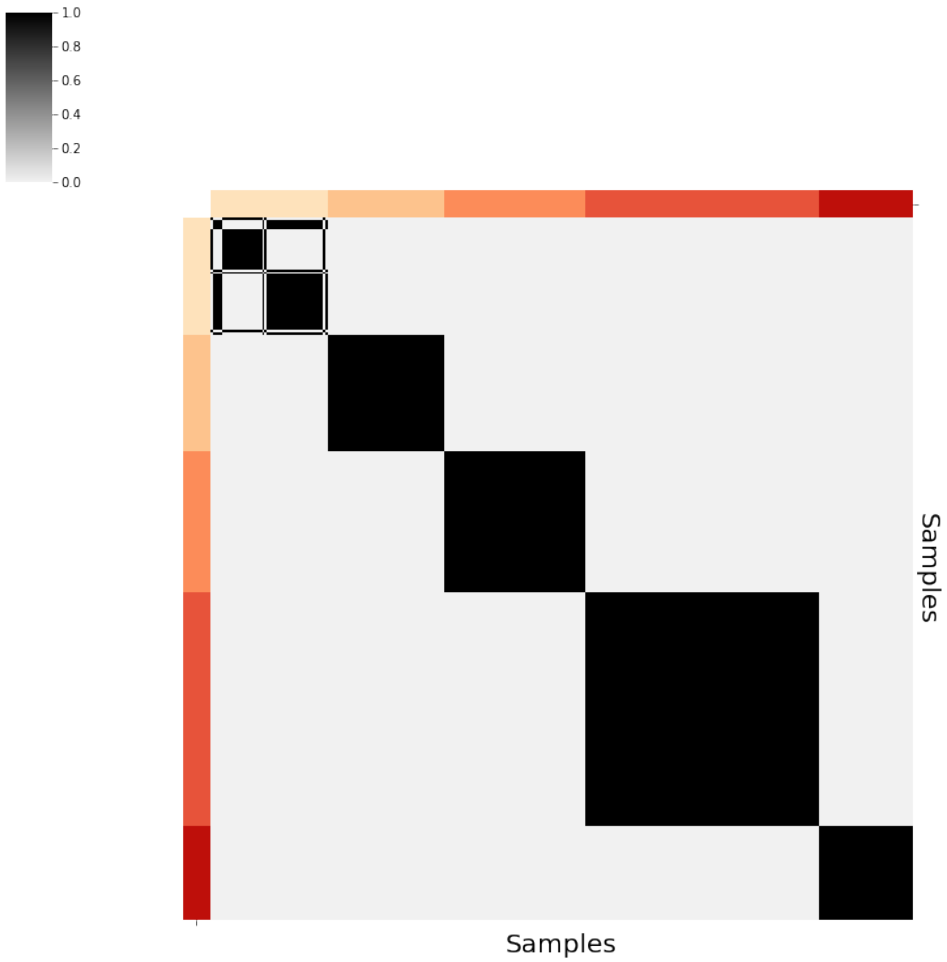
[11]:
```
vis.PlotPointEstimate(sim_data,save="plots/sim_data")
```



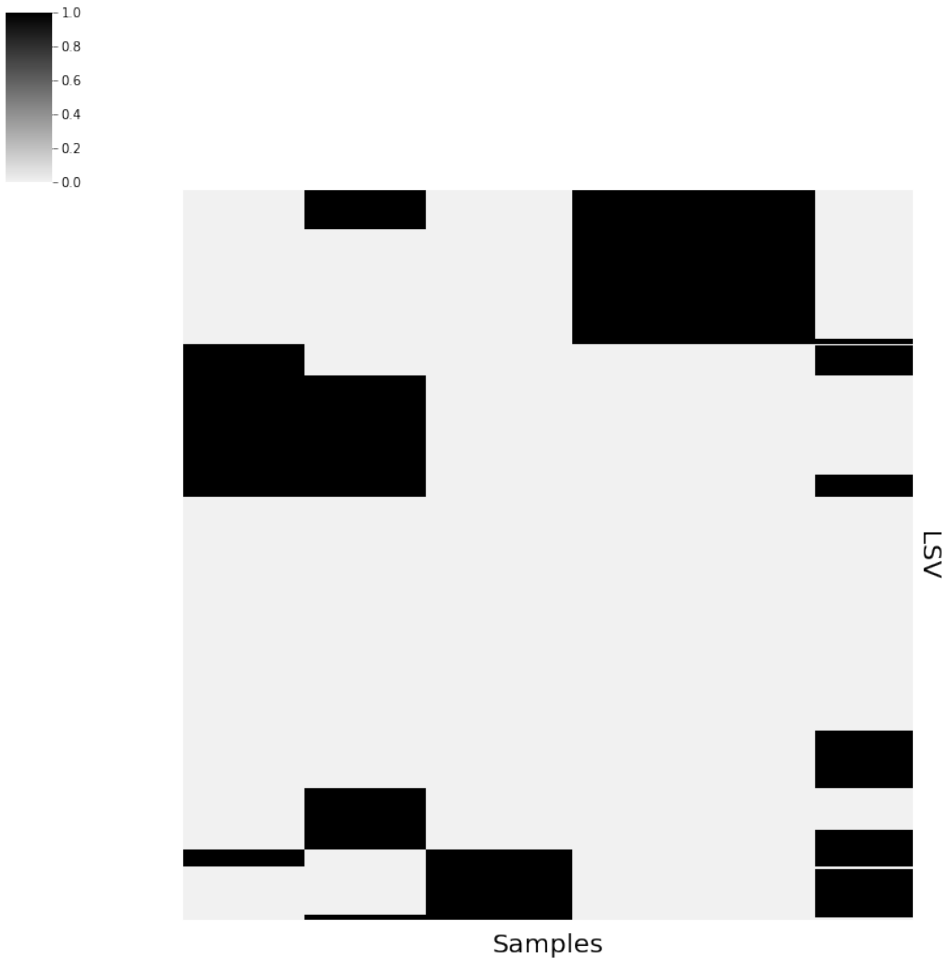
nbsphinx-code-borderwhite



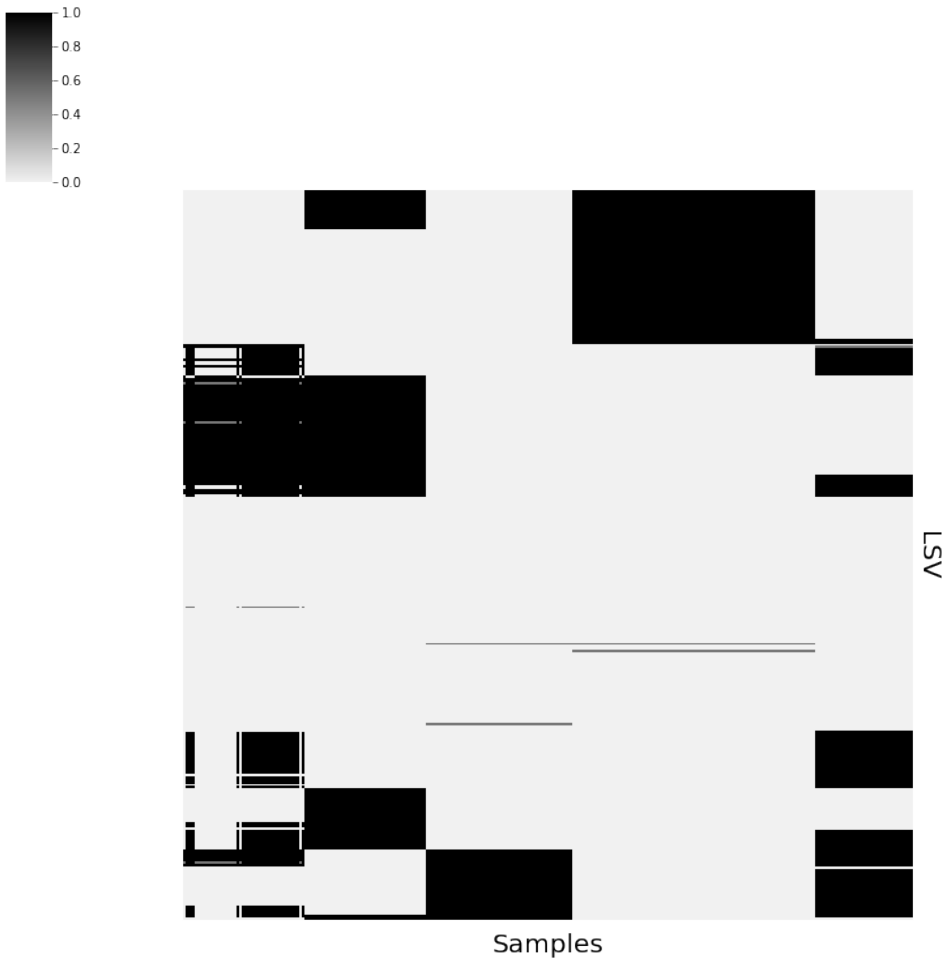
nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

[ ]:

1.4 References

BIBLIOGRAPHY

[Vaquero-Garcia16] Vaquero-Garcia *et al.* 2016, *A new view of transcriptome complexity and regulation through the lens of local splicing variations*, eLife.

PYTHON MODULE INDEX

C

INDEX

Symbols

C

D

E

G

H

L

M

O

P

S

V