# *Clumppling* Manual

V.0.1.0 (Beta Version)

Xiran Liu

Feedback/Question should be directed to: xiranliu@stanford.edu

June 12, 2023

## Contents

# 1 Introduction

*Clumppling* is a new method proposed for aligning multiple clustering replicates of population structure analysis. Comparing to exiting methods like *Clumpp* [2], *Clumpak* [3] and *Pont* [1], it provides a better tradeoff between alignment quality and computational efficiency, as well as new functionalities that are not available in those methods.

# 2 Quick Start

If you are familiar with Python and command line tools, then simply follow the steps given in this section to get a quick start on using *Clumppling*. If you are not familiar with these, do not worry. Detailed instructions are provided in the following sections.

1. Have Python 3 installed on your system (if not yet). Make sure it is callable via `python` from the command-line shell. You may check this by `python --version` to see if the expected Python 3 version is being used.

2. (Otional,recommended) Install `conda`.

3. (Otional,recommended) Create and activate a virtual environment

```
conda create -n clumppling-env python
conda activate clumppling-env
```

4. Install the convex optimization package with the solver GLPK.

```
pip install cvxpy[GLPK]==1.3 # or: conda install cvxpy[GLPK]==1.3 (if you
    have conda installed)
```

5. Install the package in one of the two following ways:

   (a) Download the package files from `https://github.com/PopGenClustering/Clumppling`, then navigate to its home directory "Clumppling", and run the command

   ```
   pip install -e .
   ```

   (b) Directly install the package by

   ```
   pip install git+https://github.com/PopGenClustering/Clumppling
   ```

   then download the example files and scripts from `https://github.com/PopGenClustering/Clumppling/tree/master/input` and `https://github.com/PopGenClustering/Clumppling/tree/master/scripts`.

   You should see all the dependencies of the program being installed.

6. Run the program with default parameters via the command

```
python -m clumppling -i INPUT_PATH -o OUTPUT_PATH -f INPUT_FORMAT
```

   where `INPUT_PATH` is the directory of all input files (clustering results), `OUTPUT_PATH` is a directory that does not exist and will be created to save the output files, and `INPUT_FORMAT` is the format of input, chosen from *structure*, *admixture*, *fastStructure* and *generalQ*.
   The output of the program will be saved in the specified directory.

7. To see a list of all required and optional parameters, run

```
python -m clumppling -h
```

# 3 Download and Installation

## 3.1 Install Python 3

*Clumppling* is based on Python 3, so you first need to have Python 3 installed on your system.

**Python 3 Version** Python version of at least 3.8 is recommended. The original package is written under Python 3.8.0, and has been tested under Python 3.8.8, Python 3.9.12, Python 3.10.12, and Python 3.11.4.

**Linux** Linux users can install Python 3 from the command window using the package manager. Make sure you have the root user access for the installation.
For RHEL (Red Hat Enterprise Linux) and CentOS users, you can install via

```
sudo yum install -y python3
```

For Ubuntu and Debian users, use

```
sudo apt-get install python3
```

**MacOS and Windows** MacOS and Windows users can download Python 3 from `https://www.python.org/downloads/`. Follow the links and get the corresponding installer for your system, then double-click on the installer file and follow the instructions to install Python 3.

## 3.2 Check Python Command

If you already have Python installed on your machine, you may run the following command in the command-line shell to check the current Python installations.

```
which python # or: which python3
```

On some systems with both Python 2 and Python 3 installed, command `python` starts the interactive Python 2 interpreter and `python3` starts the interactive Python 3. However, for convenience, we will only be using `python` prompt for the rest of the manual.
If your system uses `Python 3`, you can either switch all `python` commands in this manual to `python3` and switch all `pip` commands to `pip3`, or set the alias by running

```
alias python=python3
```

You can check the version of your Python via the command

```
python --version
```

Now you should be able to open a Python interpreter by typing

```
python -i
```

You will see information about the current Python version and some prompts, and your cursor should appear after `>>>`. To exit the interpreter, type the `exit()` after `>>>`.

* Note that there may be issues when using certain shells on Windows, like GitBash. The solution is to run `winpty python -i` instead, or to set the alias by `python=winpty`.

## 3.3 [Optional] Install Conda and Create Virtual Environment

We recommend creating a virtual environment using **Conda**, which will help manage separate package installations for different projects.

### 3.3.1 Install `conda`

You can download the installer for Anaconda at `https://www.anaconda.com/download` then follow the instruction to install it.

Alternatively, you may use Miniconda (`https://docs.conda.io/en/latest/miniconda`).

### 3.3.2 Create a virtual environment

Here we show how to use `conda` to manage the virtual environment.

Before getting started, make sure you have the Anaconda Python distribution installed and accessible. You may check this by

```
conda -V
```

To see a list of all of your environments, run

```
conda info --envs
```

Next, conda create -n yourenvname python=x.x anaconda

You may refer to the official document (`https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html`) for more information on the environment management with conda. In brief, you can create a virtual environment named `clumppling-env` with Python by

```
conda create -n clumppling-env python
```

Press `y` to proceed when prompted. You may use your preferred name instead of `clumppling-env` for the virtual environment. You may also specify a Python version by the `python=3.8`.

To activate the environment, run

```
conda activate clumppling-env # for conda 4.6 and later versions
source activate clumppling-env # for conda versions prior to 4.6
```

If you are prompted to run `conda init`, follow the instructions.

All the installation and program running should happen in this virtual environment. Once you are done using the program, deactivate the environment.

To deactivate the environment, run

```
conda deactivate
```

To remove the environment permanently, run

```
conda remove -n clumppling-env --all
```

### 3.3.3 Install the optimization solver "GLPK" in package `cvxpy`

Once you have installed `conda`, we suggest installing the required dependency `cvxpy` with the optimization solver "GLPK" separately to avoid getting an error of "`cvxpy.error.SolverError: The solver GLPK_MI is not installed`" when running the program on some systems. This can be done by

```
conda install cvxpy[GLPK] # or: pip install cvxpy[GLPK]
```

## 3.4 Install *Clumppling*

There are two ways to install the package.

The first is to install the package from the GitHub repository directly, then download the example files and scripts if you want to try the demo in Section 7.

The second is to download all package files to the local from GitHub, then install the package locally.

### 3.4.1 Install directly

**Install** To install *Clumppling* directly, run:

```
pip install git+https://github.com/PopGenClustering/Clumppling
```

**Download** Because direct installation would not automatically download the example files and scripts, you will need to get them separately from the folder "input" and "scripts" under `https://github.com/PopGenClustering/Clumppling`.

### 3.4.2 Install after downloading

You can also install the package after downloading the files to local.

**Download** Download the entire directory by clicking the link `https://github.com/PopGenClustering/Clumppling/archive/refs/heads/master.zip`. Unzipping the zipped file on your local machine, you should see a folder named "Clumppling-master", which contains the "README.md" file. Rename this directory "Clumppling". Note that this renaming step is recommended for the sake of notation consistency across this manual.

Alternatively, you may clone the package files from GitHub using Git. To install Git, download the installer and follow the instructions from `https://git-scm.com/downloads`. Next, navigate to the local directory you would like to put the files. You can then clone the repository by

```
git clone https://github.com/PopGenClustering/Clumppling.git
```

Under your current directory, a folder named "Clumppling" will show up, which contains all the files of the package.

**Install** Next, navigate to the directory named "Clumppling" where you see the files including "README.md", "pyproject.toml", and "setup.cfg". The package can be installed by

```
pip install -e .
```

After the `pip install` command, on your console you will see the text ending with "successfully installed" followed by a list of packages that are newly installed, including `clumppling-0.1.0`, indicating that your installation has succeeded. If are any compatibility issues, please resolve them first and then retry the installation.

## 3.5 Check usage

You should have installed the package now. If you're not sure where the package is installed, you can run

```
pip show clumppling
```

This will show you the installation location as well as basic information about the package.

To check the success of the installation, you may run the following command to see the help message for the main module `clumppling`:

```
python -m clumppling -h
```

If the package has been installed successfully, you will see the following output in your command window

```
usage: __main__.py [-h] -i INPUT_PATH -o OUTPUT_PATH -f INPUT_FORMAT [-v VIS
   ] [--cd_param CD_PARAM]
                   [--use_rep USE_REP] [--merge_cls MERGE_CLS] [--cd_default
                       CD_DEFAULT] [--plot_modes PLOT_MODES]
                   [--plot_modes_withinK PLOT_MODES_WITHINK] [--
                      plot_major_modes PLOT_MAJOR_MODES]
                   [--plot_all_modes PLOT_ALL_MODES] [--custom_cmap
                      CUSTOM_CMAP]

required arguments:
  -i INPUT_PATH, --input_path INPUT_PATH
                        path to load input files
  -o OUTPUT_PATH, --output_path OUTPUT_PATH
                        path to save output files
  -f INPUT_FORMAT, --input_format INPUT_FORMAT
                        input data format

optional arguments:
  -v VIS, --vis VIS     whether to generate visualization: 0 for no, 1 for
     yes (default)
```

```
--cd_param CD_PARAM   the parameter for community detection method (
   default 1.0)
--use_rep USE_REP     whether to use representative replicate as mode
   consensus: 0 for no (default), 1 for yes
--merge_cls MERGE_CLS
                      whether to merge all pairs of clusters to align K+1
                         and K: 0 for no (default), 1 for yes
--cd_default CD_DEFAULT
                      whether to use default community detection method (
                         Louvain): 0 for no, 1 for yes (default)
--plot_modes PLOT_MODES
                      whether to display aligned modes in structure plots
                         over a multipartite graph: 0 for no, 1 for
                      yes (default)
--plot_modes_withinK PLOT_MODES_WITHINK
                      whether to display modes for each K in structure
                         plots: 0 for no (default), 1 for yes
--plot_major_modes PLOT_MAJOR_MODES
                      whether to display all major modes in a series of
                         structure plots: 0 for no (default), 1 for
                      yes
--plot_all_modes PLOT_ALL_MODES
                      whether to display all aligned modes in a series of
                         structure plots: 0 for no (default), 1 for
                      yes
--custom_cmap CUSTOM_CMAP
                      customized colormap as a comma-separated string of
                         hex codes for colors: if empty (default),
                      using the default colormap, otherwise use the user-
                         specified colormap
```

# 4 Running *Clumppling*

The main module of the package runs the alignment algorithm on clustering outputs of the same model on the same set of individuals, namely, it runs the alignment on different membership matrices $Q : N \times K$ with the same number of rows $N$, but not necessarily the same number of columns $K$. This function takes in three required arguments and several optional ones:

- `input_path` (`-i`, required): path to the folder that contains the input files, i.e. the clustering results
- `output_path` (`-o`, required): path to a folder that the user wants to save the alignment results to, where the folder must not exist yet
- `input_format` (`-f`, required): the input data format, chosen from `structure`, `fastStructure`, `admixture`, and `generalQ`.
- `vis` (`-v`, optional): whether to generate visualization: 0 for no, 1 for yes (default)
- `cd_param` (optional): a numerical value of the resolution parameter for the community detection Louvain algorithm (default 1.0)
- `use_rep` (optional): whether to use representative replicate as mode consensus: 0 for no (default), 1 for yes
- `merge_cls` (optional): whether to merge all pairs of clusters to align K+1 and K: 0 for no (default), 1 for yes
- `cd_default` (optional): whether to use the default Louvain algorithm for community detection: 0 for no, 1 for yes (default). If no, then extra steps need to be taken to add a custom community detection function in the source code (see Section 8).

- `plot_modes`, `plot_modes_withinK`, `plot_major_modes`, and `plot_all_modes` (optional): whether to generate corresponding visualizations of the results: 0 for no, 1 for yes. If yes, the corresponding figure(s) will be generated.
  `plot_modes` (default 1): all aligned modes in structure plots over a multipartite graph, where better alignment between the modes is indicated by the darker color of the edges connecting their structure plots, and the cost of optimal alignment is labeled on each edge.
  `plot_modes_withinK` (default 0): A set of figures, one for each number of clusters, with all modes with the same number of clusters in structure plots in one figure.
  `plot_major_modes` (default 0): the major modes of each $K$ aligned in a series of structure plots.
  `plot_all_modes` (default 0) all aligned modes in a series of structure plots.

- `custom_cmap` (optional): a string that is either empty (default "") or contains a list of colors to be used for a customized colormap. If the string is empty then the default colormap will be used. The customized colormap, if provided, should be a list of colors in hex code in a comma-delimited string. An example colormap with five colors looks like `"#FF0000,#FFFF00,#00EAFF #AA00FF,#FF7F00"`. If the provided colormap does not have enough colors for all clusters, colors will be cycled.

The default function can be called by `python -m clumppling` followed by the arguments. You can provide arguments in either of the following ways:

```
python -m clumppling -i INPUT_PATH -o OUTPUT_PATH -f INPUT_FORMAT
```

or

```
python -m clumppling --input_path INPUT_PATH --output_path OUTPUT_PATH --
   input_format INPUT_FORMAT
```

or

```
python -m clumppling --input_path="INPUT_PATH" --output_path="OUTPUT_PATH"
   --input_format="INPUT_FORMAT"
```

where the `INPUT_PATH`, `OUTPUT_PATH`, and `INPUT_FORMAT` denote the user-specific arguments. You may also use a backslash at the end of a line for a line break:

```
python -m clumppling \
-i INPUT_PATH \
-o OUTPUT_PATH \
-f INPUT_FORMAT
```

Details about different input data formats supported by *Clumppling* are described in Section 5.

# 5 Input Data Format

The alignment function `clumppling` supports various input formats. It can be used with the alignment of the outputs of widely-used popular population structure inference methods, *Structure*, *fastStructure*, and *Admixture*, as well as the general output of any mixed-membership clustering method stored as membership matrices.

## 5.1 Clustering From *Structure*

To specify the input format as *Structure* results files, you need to use the following argument for the function `clumppling`

```
--input_format structure  # or -i structure
```

The `input_path` should point to the "Results" folder that stores the output files of *Structure*. This folder should be automatically generated when you run a *Structure* job. For instance, when you run *Structure* on a project with parameter set named `xxx` for 100 runs, this folder will contain files with names `xxx_run_1_f` to `xxx_run_100_f`. This input format is also the one we use for the demonstration in this manual.

### 5.2 Clustering From *Admixture*

To specify the input format as *Admixture* results files, you need to use the following argument for the function `clumppling`

```
--input_format admixture
```

The `input_path` should point to the a folder that stores the output files of *Admixture*, namely, those ending with "K.Q", where $K$ is the number of clusters, or those ending with ".indivq" (where additional individual information is available). These files store the ancestry fractions, as in membership matrices. It is fine to leave other files, if any, in the same folder. The program will ignore all files not with the corresponding extension. I.e., if there are 50 ".Q" files, then *Clumppling* will attempt to align these 50 runs of the clustering.

### 5.3 Clustering From *fastStructure*

To specify the input format as *fastStructure* results files, you need to use the following argument for the function `clumppling`

```
--input_format fastStructure
```

The `input_path` should point to the folder that stores the output files of *fastStructure*. Specifically, this folder needs to contain the output ".K.meanQ" files, where $K$ is the number of clusters for the run. The ".meanQ" contain the posterior mean of admixture proportions. If the folder contains other files, like ".meanP", ".varP", and ".varQ", the program will simply ignore them, so there is no need to filter those files out after running *fastStructure*. You simply need to ues the output directory of *fastStructure* as the `input_path` for *Clumppling*.

### 5.4 General Membership Matrices From Any Clustering

*Clumppling* also accepts input as general membership matrices, regardless of the clustering methods used for generating them. To specify such input format, which we term "general Q matrices", you need to use the following argument for the function `clumppling`

```
--input_format generalQ
```

The file format in the folder that `input_path` is pointing to should be the same as that for the input data coming from *Admixture* output, the Q files. The files should end with ".Q". There is no need to include $K$ in the file name, as the program will automatically detect the number of clusters when loading the clustering results. Each ".Q" file should contain the space-delimited membership matrix of a run. For instance, if there are 100 individuals clustered into 4 ancestries, then each row should contain 4 values, separated by space and summing up to 1, which correspond to the membership coefficients of an individual, and there should be 100 rows in the file.

## 6 Output Files

The main function `clumppling` generates outputs in a folder specified by `output_path` and zips them into a zip file "OUTPUT_PATH.zip". The folder shall include six subfolders and an output file "output.log". An "output.log" file contains all the output to the console when running *Clumppling*, which also documents the run time. The contents of the six subfolders are described in detail in the following subsections.

**Alignment pattern.** In the output files, the optimal alignment between a pair of replicates (or modes) is recorded as a space-separated group of numbers. For instance, if Replicate 1 and Replicate 2, each with five clusters, have the alignment pattern `4 2 1 5 3`, the five clusters in Replicate 2 should be matched to clusters 4, 2, 1, 5, and 3 in Replicate 1, respectively. The number of clusters in Replicate 2 will always be no less than that in Replicate 1. Because multiple clusters in Replicate 2 may be matched to the same cluster in Replicate 1,

### 6.1 The "input" folder

The "input" folder contains the following files:

- ".Q" files of membership matrices which are the input clustering results loaded and processed by *Clumppling*. The naming of the".Q" files, which are the labels of the replicates used by *Clumppling*, follows the pattern of `"1_K5R2.Q"`, where 1 is the index of the clustering results used in by *Clumppling*, 5 is the number of clusters, and 2 is the index of this replicate in all replicates with this number of cluster.
- A comma-delimited "input_files.txt" file with two fields. The first field is the name of processed ".Q" files and the second field is the corresponding name of the original input file.
- A "ind_info.txt" file if the input data format provides population labeling of the individuals. It contains auxiliary information on the individuals like identification label and population, where the order of the individuals is the same as that in the membership matrices.

## 6.2 The "alignment_withinK" folder

The "alignment_withinK" folder stores the results of all pairwise alignment of replicates with the same number of clusters. It contains one comma-delimited text file for each K value in the input. Each file contains three fields: three fields: `Replicate1-Replicate2`, `Cost`, and `Alignment`. `Replicate1-Replicate2` records the labels of the two replicates. `Cost` records the cost of optimal alignment between the two replicates. `Alignment` records the optimal alignment pattern between the two replicates in a space-separated group of numbers. Each line in the file, except for the header, corresponds to the alignment results of a pair of replicates with the specified number of clusters.

For example, the following line

```
151_K5R1-152_K5R2,0.01769820597265037,4 2 1 5 3
```

records the alignment results between replicate `151_K5R1` and replicate `152_K5R2`. The optimal alignment pattern between them is `4 2 1 5 3`, and under this optimal alignment, the cost between the aligned replicates is `0.01769820597265037`.

## 6.3 The "modes" folder

The "modes" folder contains the following files:

- ".Q" files of the consensus membership matrices of all modes detected. The naming of the".Q" files, which are the labels of the modes used by *Clumppling*, follows the pattern of `"K5M1_avg.Q"`, where 5 is the number of clusters, 1 is the index of this mode in all modes with this number of cluster, and `mean` stands for using the mean memberships as the consensus of mode. An alternative is `rep`, which tells the program to use the representative replicate as the consensus of mode.
- A comma-delimited "mode_alignments.txt" file with four fields: `Mode`, `Representative`, `Replicate`, and `Alignment`. A line in the file looks like this:

  ```
  K5M1,152_K5R2,151_K5R1,3 2 5 1 4
  ```

  `Mode` records the label of the mode, e.g., `K5M1`. `Representative` records the label of the representative replicate of the mode, e.g., `152_K5R2`. `Replicate` records the label of the focal replicate , e.g., `152_K5R1`. `Alignment` records the space-separated alignment pattern between the representative and the focal replicates.
- A comma-delimited "mode_stats.txt" file with five fields: `Mode`, `Representative`, `Size`, `Cost`, and `Performance`. `Mode` and `Representative` are the same as in "mode_alignments.txt". `Size`, `Cost`, and `Performance` record the size (number of clusters) of the mode, the averaged cost of the alignment between all pairs of clusters in the mode, and the averaged performance measure, calculated as the $H'$ similarity, of all the pairwise alignments in the mode. A lower cost and a higher performance value indicate a better alignment quality for the mode detected.
- A comma-delimited "mode_average_stats.txt" file with five fields: `K`, `Size`, `NS-Size`, `Cost`, and `Performance`. `K` is the number of clusters. `Size` is the total number of replicates. `NS-Size` is the number of replicates not belonging to singleton modes (i.e., modes with only one replicate). `Cost` is the weighted average of the cost of all modes not including the singletons, weighted by the mode size. `Performance` is the weighted average of the performance measure ($H'$ similarity) of all modes not including the singletons, weighted by the mode size.

## 6.4 The "alignment_across_K" folder

The "alignment_across_K" folder contains two files for each type of consensus membership of the mode, specified by the suffixes `"_avg"` and `"_rep"`.

- A comma-delimited "alignment_acrossK.txt" file with three fields: `Mode1-Mode2`, `Cost`, and `Alignment`. Each line in the file, except for the header, corresponds to the alignment results of a pair of modes. `Mode1-Mode2` records the labels of the two modes, where the second mode has a number of clusters no less than the first mode. `Cost` records the cost of optimal alignment between the two modes. `Alignment` records the optimal alignment pattern between the two modes in a space-separated group of numbers.
- A comma-delimited "best_pairs_acrossK.txt" file with five fields: `Best Pair`, `Cost`, `Alignment`, `Separate-Cluster Cost`, and `Separate-Cluster Alignment`. `Best Pair` is the best pair of modes between adjacent K in terms of the alignment performance, e.g., `K4M1-K5M1`. `Cost` and `Performance` record the cost and performance measure of optimal alignment between this pair of modes by merging the clusters in Mode 2 that get matched to the same cluster in Mode 1 and comparing two membership matrices of the same size after the merging. `Separate-Cluster Cost` and `Separate-Cluster Performance` record the cost and performance measurement that are calculated by comparing each pair of clusters that are matched and summing over the value for all clusters. For two or more clusters from Mode 2 that are matched to the same cluster in Mode 1, the average cost and performance of them are considered.

## 6.5 The "modes_aligned" folder

The "modes_aligned" folder contains the following files:

- ".Q" files of the consensus membership matrices of all modes that are aligned across different numbers of clusters. The best pair of modes for each pair of adjacent K values are used as anchors. The rest of the modes with the same K are aligned to the anchor according to results in "mode_alignments.txt" (in the "modes" folder), then the anchors are aligned according to results in the "alignment_across_K" folder. The naming of the".Q" files follows the pattern of `"K5M1_aligned_avg.Q"`, where the suffix `"_avg"` denotes the type of consensus membership used.
- A comma-delimited "all_modes_alignment.txt" file for each type of consensus membership of the mode specified by the suffixes `"_avg"` and `"_rep"`. A line in the file looks like this

  ```
  K5M1:4 3 2 1 5
  ```

  where before the colon is the mode, and after the colon is the permutation pattern that is used to generate the aligned mode in "modes_aligned" from the unaligned ones in the "modes" folder.

## 6.6 The "visualization" folder

The "visualization" folder contains all the output figures if the corresponding parameters are set to `T`.

- "colorbar.png" is the colorbar showing the colormap used in visualization, where each color represents a distinct cluster, indexed from 1 to $K$. The colormap may be specified by the user by setting the `custom_cmap` parameters.
- "modes_aligned_multipartite_avg.png" and "modes_aligned_multipartite_rep.png" plot the aligned modes over a multi-partite graph picturing the alignment relationships across different $K$. The modes are plotted in stacked bar plots. Better alignment between the modes is indicated by the darker color of the edges connecting their bar plots, and the cost of optimal alignment is labeled on each edge. These figures will only be generated if the parameter `plot_modes` is `1`.
- "major_modes_aligned_avg.png" and "major_modes_aligned_rep.png" plot the memberships in a series of stacked bar plots for the major modes for all $K$, ordering from the smallest $K$ to the largest. These figures will only be generated if the parameter `plot_major_modes` is `1`.
- One figure for each $K$ for each type of consensus membership, with the name following "K5_modes_avg.png". The figure plots the memberships in a series of stacked bar plots for all aligned modes of the specific $K$. These figures will only be generated if the parameter `plot_modes_withinK` is `1`.

- "modes_aligned_avg.png" and "modes_aligned_rep.png" plot the memberships in a series of stacked bar plots for all aligned modes for all $K$, ordering from the smallest $K$ to the largest and mode indexes from small to large. These figures will only be generated if the parameter `plot_modes` is `1`.

In all the figures, each color represents a cluster, where the colors are specified by the customized colormap if `custom_cmap` is provided.

# 7 Running Demonstration Examples

To run the demonstrations, make sure that your current directory is the "Clumppling" directory and that the folder with input files is put under "Clumppling/input". If you previously downloaded the data files as a zip file, unzip them to a folder with the same name. You may check your current working directory by running the command `pwd`.

The commands provided in this section can be used to run the examples. Alternatively, they can be run through the bash scripts which contain all the commands. To run the bash scripts, you need to install Bash on the system if it is not already there. The example bash scripts to run the demonstrations, as well as the default parameter files, are under the directory "Clumppling/scripts".

## 7.1 Datasets

To demonstrate the usage of this package, we use several published genetic datasets, and run clustering algorithms on those datasets to get the input to our alignment framework.

### 7.1.1 Human genotype data

The human genotype data contains genotype information of 44 individuals in the admixed population of Cape Verde and selected West African and Western European populations from the HGDP dataset [6]. *Admixture* was run 50 times independently for each $K$ value from 2 to 5, resulting in a total of 200 clustering replicates. This dataset is used to demonstrate the alignment of *Admixture* output files.

### 7.1.2 Chicken microsatellite data

The chicken microsatellite data comprises of 27 microsatellite loci genotyped in 600 individuals from 20 chicken populations [5]. We run *Structure* [4] on the full set of loci for 20 runs for each $K$ from 17 to 21, then align all the 100 replicates using *Clumppling*. This dataset is used to demonstrate the alignment of *Structure* output files.

## 7.2 Running `clumppling` on *Admixture* output files

To run align the clustering outputs on our example dataset from different runs of the program *Admixture* using the default community detection and visualization settings, you can run the following command

```
python -m clumppling \
--input_path input/capeverde \
--output_path output/capeverde_default \
--input_format admixture
```

Alternatively, you may write the command in a bash script and run the bash script. For example, when you are in the "Clumppling" directory, you can run the same example using the bash script in "run_ex1_default' as follows

```
bash scripts/run_ex1_default.sh
```

This example will generate the following outputs in the console

```
========== Parameters ==========
---------- [Required] ----------
Input path: input/capeverde
Input data format: admixture
Output path: output/capeverde_default
---------- [Methods] ----------
```

```
Using default community detection method: True
Community detection parameter: 1.0
Using a representative replicate as the mode consensus: False
Merging all possible pairs of clusters when aligning two replicates with K
   differing by one: False
----------- [Plotting] -----------
Providing customized colormap: False
Plotting aligned modes on top of a multipartitie graph: True
Plotting modes of the same K: False
Plotting major modes: False
Plotting all modes: False
---------------------------------
=================================
======= Running Clumppling =======
>>> Processing input data files and checking arguments
Time: 3.247s
>>> Aligning replicates within K and detecting modes
Time: 17.273s
>>>  Aligning modes across K
Time: 0.152s
>>> Plotting alignment results
Time: 11.139s
>>>  Zipping files
======== Total Time: 32.443s ========
```

### 7.3 Running `clumppling` on *Structure* output files

To run align the clustering outputs on our example dataset from different runs of the program *Structure* using the default community detection and visualization settings, you can run the following command

```
python -m clumppling \
--input_path input/chicken \
--output_path output/chicken_default \
--input_format structure --cd_param=1.05
```

Alternatively, you may write the command in a bash script and run the bash script. For example, when you are in the "Clumppling" directory, you can run the same example using the bash script in "run_ex2_default.sh" as follows

```
bash scripts/run_ex2_default.sh
```

This example will generate the following outputs in the console:

```
=========== Parameters ===========
----------- [Required] -----------
Input path: input/chicken
Input data format: structure
Output path: output/chicken_default
------------ [Methods] -----------
Using default community detection method: True
Community detection parameter: 1.05
Using a representative replicate as the mode consensus: False
Merging all possible pairs of clusters when aligning two replicates with K
   differing by one: False
----------- [Plotting] -----------
Providing customized colormap: False
Plotting aligned modes on top of a multipartitie graph: True
Plotting modes of the same K: False
```

```
Plotting major modes: False
Plotting all modes: False
---------------------------------
=================================
======= Running Clumppling =======
>>> Processing input data files and checking arguments
Time: 3.386s
>>> Aligning replicates within K and detecting modes
Time: 6.609s
>>>  Aligning modes across K
Time: 1.055s
>>> Plotting alignment results
Time: 182.411s
>>>  Zipping files
======== Total Time: 194.730s ========
```

### 7.4 Running `clumppling` on outputs with a non-consecutive number of clusters

To demonstrate *Clumppling*'s ability to align replicates when the number of clusters are not consecutive, we provide an additional example with a subset of runs from Example 2, with $K = 17$, 19, and 21. The command stays the same, except for the input path, and the corresponding bash script is

```
bash scripts/ex3.sh
```

and the commands in the script are

```
python -m clumppling \
--input_path="input/chicken_gapK" \
--output_path="output/chicken_gapK" \
--input_format="structure" --cd_param=1.05 --custom_cmap="#D65859,#00AAC1
   ,#01C0F6,#FDF0C4,#F1B38C,#AAD6BD,#6BB582,#B5DDF7,#AE8557,#FCEC73,#A4A569
   ,#4264AC,#A1CDB2,#DE9D5D,#D9439A,#ABB2BA,#8775B3,#B3865C,#DADDE6,#E7BDD1
   ,#FF9999"
```

## 8 Additional Customizable Functionalities

*Clumppling* also provides additional customizable functionalities, which require the modification of provided Python code files.

If you install the package via downloading the package files, you can find the code files under the "Clumppling/-clumppling" folder. If you install the package directly using git, you may locate your package files by

```
pip show clumppling
```

and then navigate to the location shown.

### 8.1 Custom community detection method

To use a custom community detection method, you need to modify the code file "custom_funcs.py" to implement the customized community detection function `cd_custom` and change the corresponding parameter in the parameter file. Once you locate the file, make changes to the following function

```
def cd_custom(G):
    """Customized community detection method (need to be modified)

    Parameters
    ----------
    G : networkx.Graph
```

```
        the similarity network of replicates, with edges weighted by
            similarity after optimal alignment

    Returns
    -------
    partition_map
        a dictionary where keys are the indices of replicates and values are
            the indices of the communities they belong to
    """

    # Please comment out the following line and customize your community
        detection method here.
    partition_map = {i:0 for i in range(G.number_of_nodes())}

    return partition_map
```

The `cd_custom` function takes in a NetworkX graph object, where nodes are replicates we want to align, and edges between each pair of replicates are negatively weighted by the optimal alignment cost between them. In general, nodes that are more densely connected (having edges with higher weights among them) should belong to the same community, while nodes that are connected by weaker edges (those with lower weights) should belong to different communities. The function should output a Python dictionary object containing the information of partitioning the nodes into communities.

The current function provided contains a dummy line of assigning all nodes (replicates, indexed starting from 0) into one community (mode, labeled 0). You should comment out this dummy line and implement your own community detection method that gives a valid partition map. You may refer to our default Louvain method implemented in the `cd_default` function in the file "funcs.py" as an example.

Next, reinstall the program by calling `pip install -e .` again, then you can run *Clumppling* with your customized community detection method for mode detection by setting the parameter `"cd_default"` to `0` in input arguments.

# References


[1] Aaron A Behr, Katherine Z Liu, Gracie Liu-Fang, Priyanka Nakka, and Sohini Ramachandran. pong: Fast analysis and visualization of latent clusters in population genetic data. *Bioinformatics*, 32(18):2817–2823, 2016.

[2] Mattias Jakobsson and Noah A Rosenberg. Clumpp: a cluster matching and permutation program for dealing with label switching and multimodality in analysis of population structure. *Bioinformatics*, 23(14):1801–1806, 2007.

[3] Naama M Kopelman, Jonathan Mayzel, Mattias Jakobsson, Noah A Rosenberg, and Itay Mayrose. Clumpak: a program for identifying clustering modes and packaging population structure inferences across k. *Molecular ecology resources*, 15(5):1179–1191, 2015.

[4] Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000.

[5] Noah A Rosenberg, Terry Burke, Kari Elo, Marcus W Feldman, Paul J Freidlin, Martien AM Groenen, Jossi Hillel, Asko Mäki-Tanila, Michele Tixier-Boichard, Alain Vignal, et al. Empirical evaluation of genetic clustering methods using multilocus genotypes from 20 chicken breeds. *Genetics*, 159(2):699–713, 2001.

[6] Paul Verdu, Ethan M Jewett, Trevor J Pemberton, Noah A Rosenberg, and Marlyse Baptista. Parallel trajectories of genetic and linguistic admixture in a genetically admixed creole population. *Current Biology*, 27(16):2529–2535, 2017.