Clumppling Software Manual

Version 1.2

Xiran Liu

October 6, 2025

Contents

[1]Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. “Inference of population structure using multilocus genotype data”. In: *Genetics* 155.2 (2000), pp. 945–959.

[2]David H Alexander, John Novembre, and Kenneth Lange. “Fast model-based estimation of ancestry in unrelated individuals”. In: *Genome Research* 19.9 (2009), pp. 1655–1664.

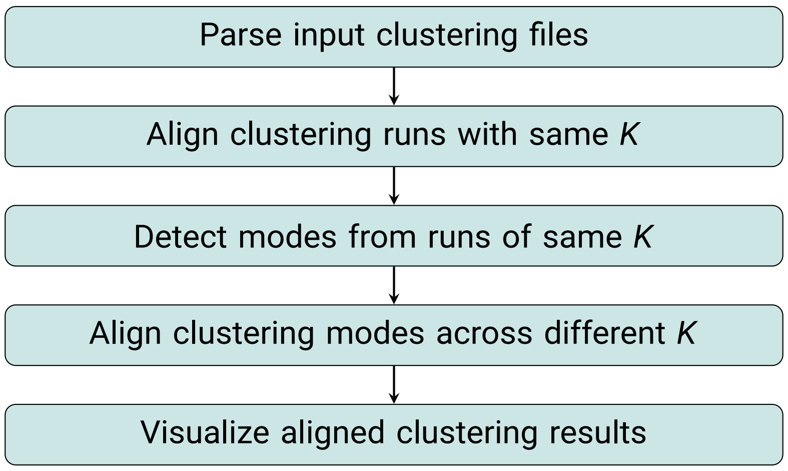
[3]Anil Raj, Matthew Stephens, and Jonathan K Pritchard. “fastSTRUCTURE: variational inference of population structure in large SNP data sets”. In: *Genetics* 197.2 (2014), pp. 573–589.

List of Code Listings

1 Introduction

This document is the software manual for `Clumppling` (Liu, Kopelman, and Rosenberg, 2024[4]). Implemented in Python, the source code and data examples are hosted on GitHub[5], and the package `clumppling` is available for installation from PyPI[6].

`Clumppling` is a program for **aligning the results of multiple clustering (ancestry inference) runs from population structure analysis** via optimization and network strategies. It aligns multiple clustering runs in the following steps:



Parse input clustering files
Align clustering runs with same K
Detect modes from runs of same K
Align clustering modes across different K
Visualize aligned clustering results

Figure 1: Workflow of `Clumppling`.

2 Quick Start

This manual provides two paths for getting started with `Clumppling`, based on your experience level.

- For experienced users familiar with Python and command-line tools, follow the quick steps in this section.

[4]Xiran Liu, Naama M Kopelman, and Noah A Rosenberg. "Clumppling: cluster matching and permutation program with integer linear programming". In: *Bioinformatics* 40.1 (2024), btad751.

[5]`https://github.com/PopGenClustering/Clumppling`

[6]`https://pypi.org/project/clumppling`

- For beginners, a detailed, step-by-step tutorial begins in the next section.
- You can also find a complete quick-start guide in the GitHub README.

To install and run the program, follow these steps.

1. Install a compatible version of Python (3.8–3.12). Ensure it's available as an appropriate python version on your command line by running `python --version` and verifying the output.

2. Install conda and create a virtual environment.

   ```
   conda create -n clumppling-env python=3.12
   ```

   Activate the created virtual environment.

   ```
   conda activate clumppling-env
   ```

3. Install the *Clumppling* package in one of the following two ways:

   (a) Directly install the package by

   ```
   pip install clumppling
   # Alternatively, you may install by:
   # pip install
   ↪  https://github.com/PopGenClustering/Clumppling/archive/master.zip
   # Or, if you have Git installed, you may also do:
   # pip install
   ↪  git+https://github.com/PopGenClustering/Clumppling
   ```

   then download the example files and scripts from the examples folder from GitHub[7].

   (b) Download all package files from GitHub repository, then navigate to its home directory titled "Clumppling" (you may need to change the automatically generated name "Clumppling-master" to "Clumppling" if you download it as a zipped file), and run the command

   ```
   pip install -e .
   ```

4. To see the usage of the program with a list of required and optional parameters, run

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

[7] https://github.com/PopGenClustering/Clumppling/tree/master/examples

5. Run the program with default parameters via the command

   ```
   python -m clumppling -i INPUT_PATH -o OUTPUT_PATH -f
   ↪  generalQ --extension .Q
   # Defaults to space-delimited files of clustering membership
   ↪  matrices. If no file extension is given, all files in
   ↪  the input path will be processed.
   ```

   where `INPUT\_PATH` is the directory of all input files (clustering results), `OUTPUT\_PATH` is a directory where the output files will be saved, and `INPUT\_FORMAT` is the format of input files, chosen from `generalQ`, `structure`, `admixture`, and `fastStructure`.

If your clustering results are not in standard output format from `Structure`[8] or `Admixture`[9], we **highly recommend** converting them to space-delimited text files. Each file should contain a membership matrix where rows represent individuals and columns represent clusters (Q_{ik} is the membership of individual i in cluster k). Then, specify `generalQ` as the input format.

3 Download and Installation

3.1 Local installation

3.1.1 Use command-line terminal

Open a command-line terminal:

- **macOS & Linux:** Use the built-in Terminal application.
- **Windows:** We recommend installing a modern terminal as the default Command Prompt (`cmd.exe`) requires manual path configuration. Some choices for the terminals are:
  - **Anaconda Prompt (Recommended):** Available after installing Anaconda (see Sections 3.1.2 and 3.1.3). To open it, search for "Anaconda Prompt" in the Start Menu.

[8]Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. "Inference of population structure using multilocus genotype data". In: *Genetics* 155.2 (2000), pp. 945–959.

[9]David H Alexander, John Novembre, and Kenneth Lange. "Fast model-based estimation of ancestry in unrelated individuals". In: *Genome Research* 19.9 (2009), pp. 1655–1664.

- **Windows PowerShell:** Built into Windows 10. Search for "Windows PowerShell" in the Start Menu.
- **Git Bash:** Available after installing Git from its installer[10]. Search for "Git Bash" in the Start Menu.

Table 1: Common command-line equivalents

Action	Unix-like Shell (Bash)	Windows (Command Prompt)
Show current directory	`pwd`	`cd`
Change directory	`cd DIRECTORY`	`cd DIRECTORY`
Go to parent directory	`cd ..`	`cd ..`
Go to Home Directory	`cd` or `cd ~`	`cd %USERPROFILE%`
List directory contents	`ls`	`dir`
Run Python interpreter	`python`	`python` or `py`

Table 1 gives some helpful command-line operations.

3.1.2 Install Python 3

This program requires **Python 3.8–3.12**. Note that Python 3.13 is not yet supported due to package incompatibilities.

- **Linux:** Install Python using your system's package manager (root access is required).
  - **For RHEL, CentOS, or Fedora:**

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

  - **For Ubuntu or Debian:**

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

- **macOS & Windows:** Download the appropriate installer from `https://www.python.org/downloads/`. Run the installer and follow the on-screen instructions.

  Note for Windows users: On the first screen of the installer, be sure to check the box labeled "Add Python.exe to PATH".

Check Your Python version (optional) If Python is already installed, you can check its version by running:

[10] `https://git-scm.com/download/win`

```
python --version
```

Note: On some systems, `python` may refer to Python 2. If this is the case for you, simply substitute `python3` for `python` and `pip3` for `pip` in all commands throughout this manual. Alternatively, you can create a temporary alias in your shell:

```
alias python=python3
```

Test the Python interpreter (optional) To ensure Python is working, start the interactive interpreter:

```
python -i
```

You should see a »> prompt. To leave the interpreter, type `exit()` and press Enter.

Note for Windows users: If you encounter issues in shells like Git Bash, you may need to run `winpty python -i` instead, or to set the alias by `python=winpty`.

3.1.3 Install Conda and Create a Virtual Environment

We strongly recommend using a **Conda** virtual environment to manage packages and prevent conflicts with other Python projects.

Install Conda Download and run the Anaconda installer from `https://www.anaconda.com/download`. After installation, you can verify your Conda version by opening a terminal (or Anaconda Prompt on Windows) and running:

```
conda -V
```

Create and Manage a Virtual Environment These steps show how to create and use a virtual environment named `clumppling-env` for this project. For more details, see the official Conda documentation.

1. **Create the environment.** Run the following command to create an environment with a specific Python version:

   ```
   conda create -n clumppling-env python=3.11 # or python=3.12
   ```

   Press y to proceed when prompted. You can replace `clumppling-env` with a name of your choice.

2. **Activate the environment.** Before installing packages or running the program, you must activate the environment:

   ```
   conda activate clumppling-env
   ```

   Note: Older versions may require `source activate clumppling-env`.

3. **Deactivate the environment.** When you are finished working, deactivate the environment to return to your base shell:

   ```
       conda deactivate
   ```

4. **List or remove environments.** To see a list of all your environments, run `conda info –envs`. To permanently delete the environment we created, run:

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

3.1.4 Install *Clumppling*

You can install the package using one of two methods. We recommend the direct installation for simplicity.

Direct installation (recommended) This method installs the package directly from an online source but does not include the example scripts.

1. **Install the package** using one of the following commands. The first is the simplest.

   ```
   # Option 1: Install from PyPI (standard)
   pip install clumppling

   # Option 2: Install directly from the GitHub repository
   pip install
    ↪  https://github.com/PopGenClustering/Clumppling/archive/master.zip

   # Option 3: Install using Git (if Git is installed)
   pip install
    ↪  git+https://github.com/PopGenClustering/Clumppling
   ```

2. **Download examples separately.** To run the demo in Section 7, you must download the contents of the “examples” folders from the GitHub repository.

Installation from source This method involves downloading all repository files first, which is useful if you want the example files or wish to modify the source code.

1. **Download the source code** using one of the following options:

   - **As a ZIP file:** Download and unzip the archive from `https://github.com/PopGenClustering/Clumppling/archive/refs/heads/master.zip`. We recommend renaming the extracted folder from "Clumppling-master" to "Clumppling".

   - **Using Git:** Clone the repository directly. This will create a folder named "Clumppling" in your current directory.

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

2. **Install the package locally.**

   (a) Navigate into the "Clumppling" directory you just downloaded:

   ```
   # Example path; replace with your own
   cd C:/Users/USERNAME/Clumppling
   ```

   (b) Install the package in editable mode (`-e`):

   ```
   pip install -e .
   ```

   A success message should list `clumppling` as one of the installed packages.

3. **Debugging (optional).** If you see an error about a `pyproject.toml` file, your `pip` may be outdated. Upgrade it and retry the installation:

   ```
   python -m pip install --upgrade pip
   ```

3.1.5 Verify the Installation

Once installed, you can check that everything is working correctly.

1. **Check package information.** To see where the package was installed, run:

   ```
   pip show clumppling
   ```

2. **View the help message.** If the package is installed correctly, this command will display the main help message:

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

3.2 Online notebook

As an alternative to a local installation, you can use the package online in a Google Colaboratory notebook: *online-notebook-for-clumppling.ipynb*[11].

To use this notebook:

1. **Sign In:** You must be signed in to a Google account to execute code.
2. **Get Your Copy:** When you open the notebook in `Colab`, a copy is automatically created in your Google Drive. All of your modifications and results will be saved to this personal copy.
3. **Save Manually (Optional):** You can also save the notebook to a specific location at any time using `File > Save a copy in Drive`.

If the installation is successfully, you will see the following help message:

```
Note: to be able to use all crisp methods, you need to install some additional
↪  packages:  {'wurlitzer', 'graph_tool', 'bayanpy'}
Note: to be able to use all crisp methods, you need to install some additional
↪  packages:  {'ASLPAw', 'pyclustering'}
Note: to be able to use all crisp methods, you need to install some additional
↪  packages:  {'wurlitzer'}
usage: __main__.py [-h] -i INPUT -o OUTPUT -f
↪  {generalQ,admixture,structure,fastStructure} [-v VIS] [--custom_cmap CUSTOM_CMAP]
↪  [--plot_type {graph,list,withinK,major,all}]
                   [--include_cost INCLUDE_COST] [--include_label INCLUDE_LABEL]
                   ↪  [--alt_color ALT_COLOR] [--ind_labels IND_LABELS]
                   ↪  [--regroup_ind REGROUP_IND]
                   [--reorder_within_group REORDER_WITHIN_GROUP] [--reorder_by_max_k
                   ↪  REORDER_BY_MAX_K] [--order_cls_by_label ORDER_CLS_BY_LABEL]
                   ↪  [--plot_unaligned PLOT_UNALIGNED]
                   [--fig_format {png,jpg,jpeg,tif,tiff,svg,pdf,eps,ps,bmp,gif}]
                   ↪  [--extension EXTENSION] [--skip_rows SKIP_ROWS]
                   ↪  [--remove_missing REMOVE_MISSING]
                   [--cd_method {louvain,leiden,infomap,markov_clustering,label_propa↩
                   ↪  gation,walktrap,custom}] [--cd_res CD_RES] [--test_comm
                   ↪  TEST_COMM] [--comm_min COMM_MIN]
                   [--comm_max COMM_MAX] [--merge MERGE] [--use_rep USE_REP]
                   ↪  [--use_best_pair USE_BEST_PAIR]

Clumppling: a tool for cluster matching and permutation program with integer linear
↪  programming

required arguments:
  -i INPUT, --input INPUT
```

[11] https://gist.github.com/xr-cc/a56458add4f7356a09f89970db40ca35

```
                        Input file path
  -o OUTPUT, --output OUTPUT
                        Output file directory
  -f {generalQ,admixture,structure,fastStructure}, --format
  ↪  {generalQ,admixture,structure,fastStructure}
                        File format

optional arguments:
  -v VIS, --vis VIS     Whether to generate figure(s): True (default)/False
  --custom_cmap CUSTOM_CMAP
                        A plain text file containing customized colors (one per line;
                        ↪  in hex code): if empty (default), using the default
                        ↪  colormap, otherwise use the user-
                        specified colormap
  --plot_type {graph,list,withinK,major,all}
                        Type of plot to generate: 'graph' (default), 'list', 'withinK',
                        ↪  'major', 'all'
  --include_cost INCLUDE_COST
                        Whether to include cost values in the graph plot: True
                        ↪  (default)/False
  --include_label INCLUDE_LABEL
                        Whether to include individual labels in the plot: True
                        ↪  (default)/False
  --alt_color ALT_COLOR
                        Whether to use alternative colors for connection lines: True
                        ↪  (default)/False
  --ind_labels IND_LABELS
                        A plain text file containing individual labels (one per line)
                        ↪  (default: last column from labels in input file, which
                        ↪  consists of columns [0, 1, 3]
                        separated by delimiter)
  --regroup_ind REGROUP_IND
                        Whether to regroup individuals so that those with the same
                        ↪  labels stay together (if labels are available): True
                        ↪  (default)/False
  --reorder_within_group REORDER_WITHIN_GROUP
                        Whether to reorder individuals within each label group in the
                        ↪  plot (if labels are available): True (default)/False
  --reorder_by_max_k REORDER_BY_MAX_K
                        Whether to reorder individuals based on the major mode with
                        ↪  largest K: True (default)/False (based on the major mode
                        ↪  with smallest K)
  --order_cls_by_label ORDER_CLS_BY_LABEL
                        Whether to reorder clusters based on total memberships within
                        ↪  each label group in the plot: True (default)/False (by
                        ↪  overall total memberships)
  --plot_unaligned PLOT_UNALIGNED
                      Whether to plot unaligned modes (in a list): True/False (default)
  --fig_format {png,jpg,jpeg,tif,tiff,svg,pdf,eps,ps,bmp,gif}
                        Figure format for output files (default: tiff)
  --extension EXTENSION
                        Extension of input files
  --skip_rows SKIP_ROWS
                        Skip top rows in input files
  --remove_missing REMOVE_MISSING
                        Remove individuals with missing data: True (default)/False
  --cd_method
  ↪  {louvain,leiden,infomap,markov_clustering,label_propagation,walktrap,custom}
```

```
                      Community detection method to use (default: louvain)
  --cd_res CD_RES       Resolution parameter for the default Louvain community
  ↪  detection (default: 1.0)
  --test_comm TEST_COMM
                      Whether to test community structure (default: True)
  --comm_min COMM_MIN   Minimum threshold for cost matrix (default: 1e-6)
  --comm_max COMM_MAX   Maximum threshold for cost matrix (default: 1e-2)
  --merge MERGE         Whether to merge two clusters when aligning K+1 to K (default:
  ↪  True)
  --use_rep USE_REP     Use representative modes (alternative: average): True
  ↪  (default)/False
  --use_best_pair USE_BEST_PAIR
                      Use best pair as anchor for across-K alignment (alternative:
                      ↪  major): True (default)/False
```

4 Running the Program

The main module, `clumppling`, aligns clustering results from multiple runs of the same model. It processes membership matrices (Q) that have the same number of individuals (N rows) but may have different numbers of clusters (K columns).

4.1 Arguments

The function requires **three** arguments and accepts several optional ones.

Table 2: Program Arguments

Argument	Description	Data Type
Required Arguments		
`-i, −input`	Input file path.	`string` (required)
`-o, −output`	Output file directory.	`string` (required)
`-f, −format`	File format. {`generalQ, admixture, structure, fastStructure`}	`string` (required)
Input File Options		

Table 2 – continued from previous page

Argument	Description	Data Type
`−extension`	Extension of input files (e.g., `.qopt`). If not specified, all files under the input path are used.	`string` (default: empty)
`−skip_rows`	Number of top rows to skip in input files.	`int` (default: 0)
`−remove_missing`	Remove individuals with missing data.	`boolean` (default: True)
`−ind_labels`	A plain text file containing individual labels (one per line).	`string` (default: empty)
Algorithm Options		
`−cd_method`	Community detection method. {`louvain`, `leiden`, `custom` etc.}	`string` (default: louvain)
`−cd_res`	Resolution parameter for Louvain community detection.	`float` (default: 1.0)
`−test_comm`	Whether to test community structure.	`boolean` (default: True)
`−comm_min`	Minimum threshold for the cost matrix.	`float` (default: 1e-6)
`−comm_max`	Maximum threshold for the cost matrix.	`float` (default: 1e-2)
`−merge`	Merge two clusters when aligning K+1 to K.	`boolean` (default: True)
`−use_rep`	Use representative modes instead of averaging.	`boolean` (default: True)
`−use_best_pair`	Use the best pair as an anchor for across-K alignment.	`boolean` (default: True)
Plotting Options		
`-v, −vis`	Whether to generate figures.	`boolean` (default: True)
`−plot_type`	Type of plot to generate. {`graph`, `list`, `withinK`, `major`, `all`}	`string` (default: graph)
`−custom_cmap`	A plain text file containing customized colors (one per line; in hex code)	`string` (default: empty)
`−include_cost`	Include cost values in the graph plot.	`boolean` (default: True)
`−include_label`	Include individual labels in the plot.	`boolean` (default: True)
`−regroup_ind`	Regroup individuals so that those with the same labels stay together.	`boolean` (default: True)

Table 2 – continued from previous page

Argument	Description	Data Type
`–reorder_within_group`	Reorder individuals within each label group in the plot based on their memberships of the cluster with largest total membership in the label group (when labels are available).	`boolean` (default: True)
`–reorder_by_max_k`	Reorder individuals based on clusters in the major mode with the largest K; otherwise using the major mode with the smallest K.	`boolean` (default: True)
`–order_cls_by_label`	Reorder clusters based on total memberships within each label group.	`boolean` (default: True)
`–plot_unaligned`	Plot unaligned modes (in a list view) for comparison (when `–plot_type=all` or `–plot_type=list`).	`boolean` (default: False)
`–fig_format`	Format of output figures. {`png`, `jpg`, `jpeg`, `tif`, `tiff`, `pdf`, `svg`, `eps`, `ps`, `bmp`, `gif`}	`string` (default: tiff)

4.2 Commands for running `clumppling`

The main program is run using `python -m clumppling` followed by its arguments. The three required arguments can be provided in several ways.

For example, using the short-form flags:

```
python -m clumppling -i path/to/input/ -o path/to/output/ -f
 ↪  generalQ
```

Or, using the long-form flags with spaces:

```
python -m clumppling --input path/to/input/ --output
 ↪  path/to/output/ --format generalQ
```

For readability, you can also use a backslash (\) to break a long command across multiple lines:

```
python -m clumppling \
    -i path/to/input/ \
    -o path/to/output/ \
    -f generalQ \
    --vis True \
    --plot_type graph
```

4.3 Algorithm Options Explained

4.3.1 Adjusting community detection resolution with –`cd_res`

The community detection process involves a trade-off: a higher resolution identifies more, smaller modes, which increases within-mode similarity but may result in too many singletons (modes with only one clustering run). Conversely, a lower resolution creates fewer, larger modes but may group dissimilar runs together.

When using the default Louvain algorithm[12], you can control the "resolution" parameter with the –`cd_res` parameter (set to 1.0 by default). We suggest performing a few test runs to find a value that balances the number of modes with their internal consistency. For complex datasets, consider using a resolution higher than the default of 1.0 to generate more fine-grained modes. If the default setting produces noisy results (e.g., too many singletons—modes with a single run—or dissimilar runs within a mode), exploring a range of values for the resolution parameter is highly recommended.

4.3.2 Aligning Across K with –`merge`

When aligning modes with numbers of clusters differing by one (e.g., K and $K+1$), two approaches are available:

- **Merge approach (–`merge True`, default):** This method, similar to that in PONG[13], exhaustively tests all possible ways to merge two clusters from the $K+1$ run to match a single cluster in the K run. By default, if K values

[12]Vincent D. Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* P10008 (2008).

[13]Aaron A Behr et al. "pong: Fast analysis and visualization of latent clusters in population genetic data". In: *Bioinformatics* 32.18 (2016), pp. 2817–2823.

are consecutive, this option is used. While it aims for a "cluster-merging optimality," it is computationally intensive and becomes infeasible when the number of clusters differs by more than one.

- **Direct approach (–`merge False`):** This is `Clumppling`'s standard method when the different between *K* is more than one. It aligns clusters from one run to the most similar clusters in another, even if it means matching multiple clusters to a single target. This method is efficient and works across any range of *K* values.

4.3.3 Customizing community detection

For advanced use cases, you can implement your own community detection algorithm instead of using the built-in methods. This is done by modifying a single Python file in the source code. 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, you may locate your package files by

```
pip show clumppling
```

and then navigate to the location shown.

1. **Implement your algorithm.** Open the file "detectMode/custom.py" in the source code. Inside, you will find a function named `cd_custom`. The file looks like this:

    - **Input:** The function receives a weighted adjacency matrix where higher values indicate greater similarity between clustering runs.
    - **Task:** Your code should partition the graph represented by this matrix into communities.
    - **Output:** The function must return a Python list of community labels, where each element in the list corresponds to a node (a clustering run).

    The file contains example code for the Markov clustering algorithm, the infomap algorithm, as well as a dummy line that outputs each node in its own community (0-indexed). You should comment out or replace the existing examples with your own implementation.

```python
def cd_custom(adj_mat: np.ndarray) -> List[int]:

    n_nodes = adj_mat.shape[0]
    # Please comment out the following lines and customize your
     ↪  method here.

    # ## Dummuy placeholder
    # communities = list(range(n_nodes))
    # logger.warning(f"Using dummy custom community detection
     ↪  method. Output will be each node in its own community
     ↪  (mode).")

    # ## Example 1: use Markov clustering
    import markov_clustering
    result = markov_clustering.run_mcl(adj_mat,
     ↪  pruning_threshold=0.2)
    coms = markov_clustering.get_clusters(result)
    communities = [-1] * n_nodes
    for cluster_idx, nodes in enumerate(coms):
        for node in nodes:
            communities[node] = cluster_idx

    ## Example 2: use infomap
    # import networkx as nx
    # from cdlib import algorithms
    # G = nx.from_numpy_array(adj_mat*100)  # Scale the
     ↪  adjacency matrix
    # G.remove_edges_from(nx.selfloop_edges(G))
    # coms = algorithms.infomap(G)
    # logger.info(f"Custom method (example 2. infomap clustering
     ↪  with adjacency matrix scaled by 100) detected
     ↪  {len(coms.communities)} communities.")
    # communities = [-1] * n_nodes
    # for cluster_idx, nodes in enumerate(coms.communities):
    #     for node in nodes:
    #         communities[node] = cluster_idx

    return communities
```

Listing 1: Custom community detection function.

2. **Re-install and run the program.** After saving your changes to `custom.py`, you may re-install the package in editable mode to make your custom function available. Navigate to the root `Clumppling` directory (where the `pyproject.toml` file is located) and run:

   ```
   pip install -e .
   ```

   Alternatively, simply navigate to the root `Clumppling` directory and call the main function from there.

   You can now run *Clumppling* with your custom method by setting the `–cd_method` argument to `custom`.

   ```
   python -m clumppling -i path/to/input/ -o path/to/output/
    ↪  -f generalQ --cd_method custom
   ```

4.3.4 Other algorithm parameters

- `–test_comm`: When enabled (default), this option first performs a statistical test for community structure before proceeding with mode detection. However, it can happen that, for example, even when the statistical test indicates significant community structure, the detected structure may be too fine-grained if interest lies in broader-level structural differentiation. Alternatively, users can disable this test. Users can also control the mode detection sensitivity using two thresholds:

  `–comm_min` (default: 1e-4): Sets the maximum alignment cost allowed within a single mode. If all pairwise costs are below this value, all runs will be grouped into one mode.

  `–comm_max` (default: 1e-2): Sets the minimum alignment cost required for grouping. If all pairwise costs are above this value, every run will become its own singleton mode.

- `–use_rep`: By default, `Clumppling` aligns entire modes using a single, representative run from each mode. Setting this to `False` will instead use the average of all runs in the mode to represent and generate the membership matrix of the mode.
- `–use_best_pair`: When aligning modes across different values of K, this option (default) uses the single most similar pair of runs as an "anchor." Setting it to `False` will instead use the major mode (the one with the most runs) as the anchor.

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`, `Admixture`, `fastStructure`, as well as the general output of any mixed-membership clustering method stored as membership matrices. Please note that `Clumppling` assumes that input files are error-free, such as having no individuals with invalid memberships. If an individual is missing membership, by default it will be filtered out during the parsing of input files (`-remove_missing=T`). Other errors (e.g., invalid numeric number) in the input files may lead to issues when loading the data. To prevent this, we recommend users review and clean their data files in advance.

5.1 Clustering runs from `Structure`[14]

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

```
--format structure  # or -f structure
```

The `--input` 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 a parameter set named `xxx` for 100 runs, this folder will contain files with names `xxx_run_1_f` to `xxx_run_100_f`. Specifically, `Clumppling` will extract the membership coefficients from the section "Inferred ancestry of individuals:" in these files.

An example of such file contains lines like the following

```
Inferred ancestry of individuals:
        Label (%Miss) Pop:  Inferred clusters
  1   01_1    (0)    1 :  0.003 0.001 0.002 0.983 0.002 0.010
  2   01_2    (0)    1 :  0.002 0.001 0.002 0.984 0.002 0.009
  3   01_3    (3)    1 :  0.003 0.001 0.002 0.982 0.003 0.009
  4   01_4    (0)    1 :  0.001 0.001 0.009 0.963 0.019 0.007
  5   01_5    (0)    1 :  0.003 0.001 0.002 0.967 0.014 0.013
```

[14] Pritchard, Stephens, and Donnelly, see n. 8.

5.2 Clustering runs from `Admixture`[15]

To specify the input format as `Admixture` result files, you need to use the following argument

```
--format admixture  # or -f admixture
```

The `--input` should point to a folder that stores the output files of `Admixture`, namely, those ending with "K.Q", where *K* is the number of clusters. These `Admixture` output files record the inferred ancestry fractions in membership matrices. It is fine to leave other files (e.g., the ".P" files), if any, in the same folder. The program will ignore all files not with the corresponding extension. E.g., if there are 50 ".Q" files, then `Clumppling` will attempt to align these 50 runs of the clustering. An example of such file contains lines like the following

```
0.00001 0.99998 0.00001
0.00001 0.99998 0.00001
0.00001 0.99998 0.00001
0.029375 0.970615 0.00001
0.00001 0.99998 0.00001
```

It can also accept the manipulated result files ending with ".indivq" in which additional individual information is stored in the same manner as in `Structure` files, i.e., the additional individual information is available in the leading 4 columns (space-delimited, before the column with ":") before the membership coefficients, representing the individual index, the individual label, the percentage of missingness, and the population label. An example of such file contains lines like the following

```
1 XXX1 (0) 1 : 0.000010 0.999980 0.000010
2 XXX2 (0) 1 : 0.000010 0.999980 0.000010
3 XXX3 (0) 1 : 0.000010 0.999980 0.000010
4 XXX4 (0) 1 : 0.029375 0.970615 0.000010
5 XXX5 (0) 1 : 0.000010 0.999980 0.000010
```

[15]Alexander, Novembre, and Lange, see n. 9.

5.3 Clustering runs from `fastStructure`[16]

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

```
--format fastStructure
```

The `--input` 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" contains the posterior mean of admixture proportions. Except for the difference in file extension, the actual structure of the files should be the same as those of `Admixture`. If the folder contains other files, like ".meanP", ".varP", and ".varQ", the program will ignore them if user specifies `--extension .meanQ`, so there is no need to filter those files out after running *fastStructure*. You simply need to use the output directory of `fastStructure` as the `--input`.

5.4 General membership matrices from any clustering tools

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

```
--format generalQ
```

The file format in the folder that `--input` points to **should be the same as that for the input data coming from `Admixture` output, the Q files**. The files by default end with ".Q". Otherwise, user will need to sepcify the file extension explicitly through `--extension`. 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 3 ancestries, then each row should contain 3 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.

[16] Anil Raj, Matthew Stephens, and Jonathan K Pritchard. "fastSTRUCTURE: variational inference of population structure in large SNP data sets". In: *Genetics* 197.2 (2014), pp. 573–589.

5.5 Optional custom colors and population labels

When users choose to use a **custom color map** for visualization, they may provide a file with color codes, one color per line, using the `--custom_cmap` argument. If the number of clusters is larger than the number of colors provided, colors will be recycled. An example custom color file looks like

```
#D65859
#00AAC1
#01C0F6
#FDF0C4
#F1B38C
#AAD6BD
#6BB582
#B5DDF7
```

To provide **population labels** for individuals, use the `--ind_labels` argument to specify a text file containing one label per line. The order of labels in this file must match the order of individuals in the membership matrices.

If either population labeling information can be extracted from the input files, or labels are provided separately, the following plotting arguments will be available for activation: `--include_label`, `regroup_ind`, `--reorder_within_group`, `--reorder_by_max_k`, and `--order_cls_by_label`.

6 Output Files

The program generates all output in the folder specified by `--output` and creates a compressed archive of this folder named "output.zip". The main output folder contains several subfolders and a log file.

Understanding alignment patterns Throughout the output files, an alignment between two runs (termed "replicates") is represented as a space-separated sequence of numbers. For example, an alignment pattern of 4 2 1 5 3 between Replicate 1 (with 5 clusters) and Replicate 2 (with 5 clusters) means that clusters 1-5 of Replicate 2 map to clusters 4, 2, 1, 5, and 3 of Replicate 1, respectively.

6.1 The `input` folder

This folder contains processed versions of your input data.

- **`*.Q` files:** The membership matrices used for alignment, renamed for internal consistency (e.g., `1_K5R2.Q`, where K5 indicates 5 clusters and R2 indicates the second replicate for that K).
- **`input_meta.txt`:** A comma-delimited file of input clustering runs' meta information. Three fields correspond to the original input filename (and path), the new `.Q` filename (and path), and the number of clusters for each clustering run, with one clustering run per line.
- **`input_labels.txt` (optional):** A comma-delimited file of individuals' labeling information. If your input format includes population labels, this file stores that information for each individual, in the same order as the membership matrices. Three fields correspond to the individual's index, ID, and population label, with one individual per line.
- **`ind_labels_grouped.txt` (optional):** If population labels are provided and regrouped (so that individuals with the same label stay together), the new (reordered) labels will be saved to this file, with one label per line.
- **`ind_labels_indices.txt` (optional):** If population labels are provided and regrouped (so that individuals with the same label stay together), the indices of individuals in original ordering (0-based) will be saved to this file, with one index per line.

6.2 The `alignment_withinK` folder

This folder contains the pairwise alignment results for all runs with the same number of clusters (K). For each value of K, a comma-delimited text file with the alignment cost and pattern for each pair of runs with the same K. An example file "K3.txt" looks like

```
Replicate1-Replicate2,Cost,Alignment
51_K3R1-52_K3R2,2.6195234708117053e-09,2 3 1
52_K3R2-51_K3R1,2.6195234708117053e-09,3 1 2
51_K3R1-53_K3R3,4.98453557895081e-09,3 1 2
53_K3R3-51_K3R1,4.98453557895081e-09,2 3 1
```

6.3 The `modes` folder

This folder contains results from the mode detection analysis.

- **`*.Q` files:** Consensus membership matrices for each detected mode (e.g., `K5M1_avg.Q`). The suffix indicates whether the consensus was derived from the mean memberships (`_avg`) or a representative run (`_rep`).
- **`mode_alignment.txt`:** A comma-separated file detailing how each replicate aligns to its mode's representative. Example:

```
Mode,Representative,Replicate,Alignment
K2M1,43_K2R43,1_K2R1,1 2
K2M1,43_K2R43,2_K2R2,2 1
K2M1,43_K2R43,3_K2R3,1 2
K2M1,43_K2R43,4_K2R4,1 2
```

- **`mode_stats.txt`:** A comma-separated file with statistics for each mode, including its size, average internal alignment cost, and performance (H' similarity). Lower cost and higher performance indicate a more coherent mode. Example:

```
Mode,Representative,Size,Cost,Performance
K2M1,43_K2R43,50,8.043782894508126e-14,0.9999997631447929
K3M1,63_K3R13,50,3.0678994759509533e-09,0.9999591846944903
K4M1,127_K4R27,39,0.0007341008166961233,0.978889151289992
K4M2,141_K4R41,11,0.04944348905103199,0.7918294905930834
```

- **`mode_average_stats.txt`:** A comma-separated file summarizing the statistics for each K, including the total number of runs, the total number of runs in non-singleton modes, and the weighted average cost and performance across all non-singleton modes. Example:

```
K,Size,NS-Size,Cost,Performance
2,50,50,8.043782894508126e-14,0.9999997631447929
3,50,50,3.0678994759509533e-09,0.9999591846944903
4,50,50,0.004099699249973099,0.965964174734301
5,50,50,0.039680709479559026,0.8158997733430288
```

6.4 The `alignment_acrossK` folder

This folder stores the alignment results between modes with different values of *K*. The suffix indicates whether the consensus was derived from the mean memberships (`_avg`) or a representative run (`_rep`).

- **`alignment_acrossK.txt`:** A comma-separated file containing the pairwise alignment cost and pattern for all pairs of modes. Example:

```
Mode1-Mode2,Cost,Alignment
K4M1-K5M1,0.005484038961324244,3 4 1 2 1
K4M2-K5M1,0.03014841177342146,4 2 2 3 1
K4M1-K5M2,0.00259844052230715,4 2 3 1 3
K4M2-K5M2,0.05803251212693439,2 3 4 2 1
```

- **`best_pairs_acrossK.txt`**: A comma-separated file identifying the best-aligned pair of modes between adjacent values of *K* (when using argument `--use_best_pair T`) and the alignment performances. Example:

```
Best Pair,Cost,Performance,Separate-Cluster
 ↪  Cost,Separate-Cluster Performance
K4M1-K5M2,0.00259844052230715,0.9490250990946805,0.001266⌋
 ↪  5292562767366,0.9644116696615768
K3M1-K4M1,4.257996082220217e-06,0.9979365087637162,3.7145⌋
 ↪  377588562873e-06,0.9980726863880374
K2M1-K3M1,3.504855027953341e-05,0.9940798183913385,1.7524⌋
 ↪  27513976679e-05,0.9958137994386596
```

- **`major_pairs_acrossK.txt`**: A comma-separated file showing the pair of major modes between adjacent values of *K* (when using argument `--use_best_pair F`) and the alignment performances. Example:

```
Major Pair,Cost,Performance,Separate-Cluster
 ↪  Cost,Separate-Cluster Performance
K4M1-K5M1,0.005484038961324244,0.9259457026140667,0.00213⌋
 ↪  7667259653589,0.9537650861398705
K3M1-K4M1,4.257996082220217e-06,0.9979365087637162,3.7145⌋
 ↪  377588562873e-06,0.9980726863880374
```

```
K2M1-K3M1,3.504855027953341e-05,0.9940798183913385,1.7524
↪  27513976679e-05,0.9958137994386596
```

6.5 The `modes_aligned` folder

This folder contains the final, globally aligned modes.

- **`*.Q` files:** The consensus membership matrices for each mode after being permuted to align across all values of *K*.
- **`all_modes_alignment.txt`:** A comma-separated file showing the permutation pattern applied to each mode to achieve the final alignment. Example:

```
K2M1:1 2
K3M1:2 1 3
K4M1:1 3 2 4
K4M2:2 4 3 1
K5M1:3 1 4 2 5
```

6.6 The `visualization` folder

This folder contains all figures generated by the program, controlled by the `--plot_type` argument. The suffix indicates whether the consensus was derived from the mean memberships (`_avg`) or a representative run (`_rep`). Figures can be chosen to be of any of the following formats (set by `--fig_format`): PNG, JPG, JPEG, TIF, TIFF (default), SVG, PDF, EPS, PS, BMP, GIF.

- **`all_modes_graph`:** The main visualization, showing modes as structure plots connected in a multi-partite graph over a grid. Each row in the grid corresponds to a *K* value, and each column corresponds to a mode. Darker edges indicate better alignment, with alignment costs labeled on the edges. This is generated by `--plot\_type graph`.

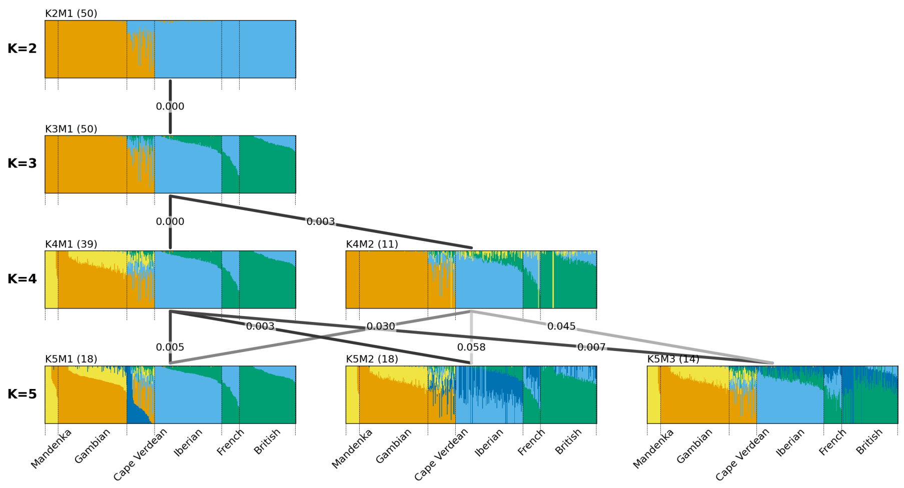


Figure 2: **`all_modes_graph`:** All modes aligned on a multi-partite graph.

- **`alignment_pattern_graph`:** Alignment patterns shown as scatter plots with connections. Each colored marker represents a cluster, and clusters from the same mode form a block. The layout of modes follows that in the `all_modes_graph`. Clusters between modes with adjacent *K* values are connected if they are aligned and not directly matched (e.g., not Cls.1 to Cls.1). Line styles alternate to better distinguish between modes.

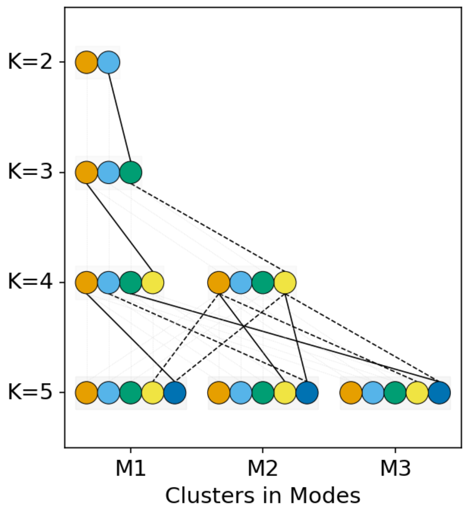


Figure 3: **`alignment_pattern_graph`:** The alignment pattern in a graph layout resembling that in the `all_modes_graph`.

- **`colorbar`:** A legend for the colormap used in the plots.

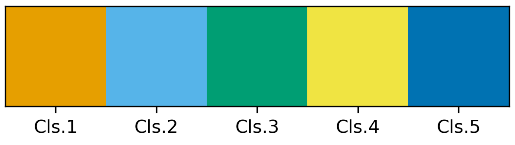


Figure 4: **`colorbar`:** The colormap used.

- **`major_modes`:** A series of structure plots showing only the major modes for each *K*. Generated with `--plot_type major`.

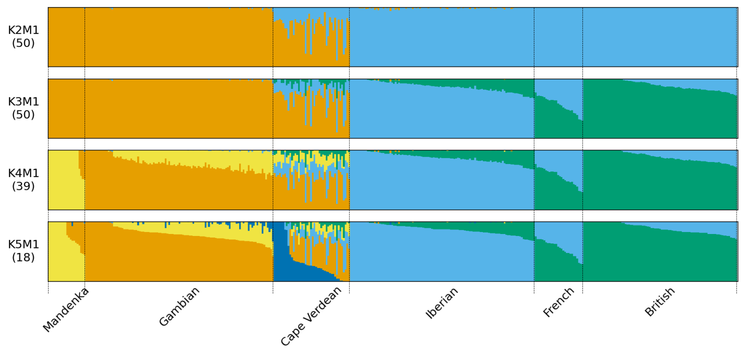


Figure 5: **`major_modes`:** All major modes aligned.

- **`all_modes_list`:** A series of structure plots for all aligned modes across all *K*, in a list view. Generated with `--plot_type list`.

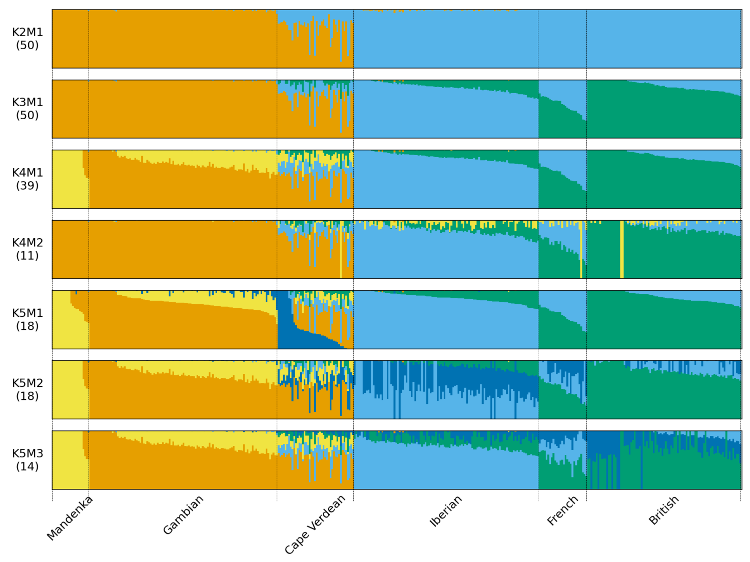


Figure 6: **`all_modes_list`:** All modes aligned in a list view.

- **`alignment_pattern_list`:** Alignment patterns shown as connected markers across modes, in a list view. Each colored marker represents a cluster, each mode takes over a row, and clusters between different modes are connected if they are aligned.

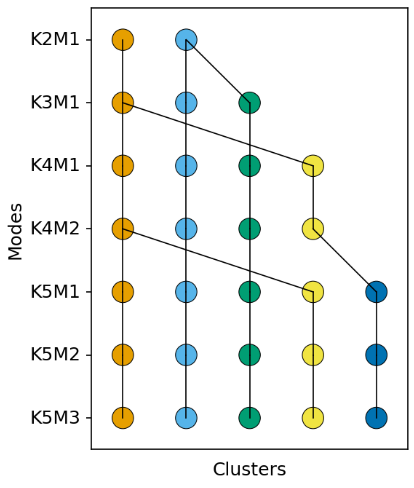


Figure 7: **`alignment_pattern_list`:** The alignment pattern in a list view resembling that in the `all_modes_list`.

- **`K*_modes`:** Figures showing all aligned modes, but separated by each value of *K*. Generated with `--plot_type withinK`.

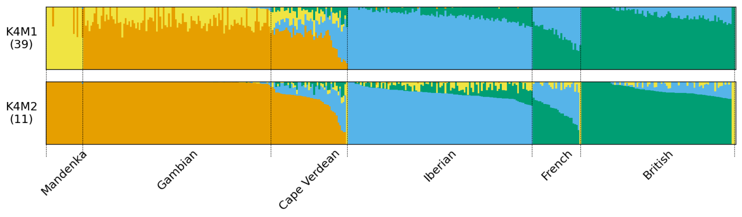


Figure 8: **`K4_modes`:** All $K = 4$ modes aligned.

7 Examples

7.1 Cape Verde data (`Admixture`/general Q)

There are 399 individuals in the dataset, including 44 from Cape Verde and the remainder from African and European populations. `Admixture` David H Alexander, John Novembre, and Kenneth Lange. "Fast model-based estimation of ancestry in unrelated individuals". In: *Genome Research* 19.9 (2009), pp. 1655–1664 was run 50 times independently for each *K* value from 2 to 5, resulting in a total of 200 clustering replicates. This dataset was analyzed in Verdu et al. 2017[17]. This dataset is used to demonstrate the alignment of `Admixture` (or general Q) output files with population labels.

Commands:

```
# !/bin/bash

# default run with default: louvain mode detection,
↪  representative Q matrices for modes, and alignment across-K
↪  using the best mode pair
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_output \
-f admixture \
--extension .indivq \
--ind_labels examples/capeverde_ind_labels.txt
```

[17]Paul Verdu et al. "Parallel trajectories of genetic and linguistic admixture in a genetically admixed creole population". In: *Current Biology* 27.16 (2017), pp. 2529–2535.

```
# run with average Q matrices for modes and alignment across-K
↪  using the major mode pair
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_avg_output \
-f admixture \
--use_rep F \
--use_best_pair F \
--extension .indivq \
--ind_labels examples/capeverde_ind_labels.txt

# run with all plot types
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_output \
-f admixture \
--extension .indivq \
--plot_type all \
--ind_labels examples/capeverde_ind_labels.txt

# run with figures in .png format
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_output \
-f admixture \
--use_rep F \
--use_best_pair F \
--extension .indivq \
--plot_type graph \
--fig_format png \
--ind_labels examples/capeverde_ind_labels.txt

# run with customized mode detection
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_custom_output \
-f admixture \
--use_rep F \
--use_best_pair F \
--extension .indivq \
--cd_method custom \
--ind_labels examples/capeverde_ind_labels.txt

# run without reordering individuals within each label group,
 ↪  and plot unaligned modes as well
python -m clumppling \
-i examples/capeverde \
-o examples/capeverde_output_unordered_w_unaligned \
-f admixture \
--use_rep F \
--use_best_pair F \
--extension .indivq \
--reorder_within_group F \
--plot_type all \
--plot_unaligned T \
--ind_labels examples/capeverde_ind_labels.txt
```

The output files and figures are those shown as examples in Section 6.

7.2 Chicken data (`Structure`)

The chicken microsatellite data comprises 27 microsatellite loci genotyped in 600 individuals from 20 chicken populations Noah A Rosenberg et al. “Empirical evaluation of genetic clustering methods using multilocus genotypes from 20 chicken breeds”. In: *Genetics* 159.2 (2001), pp. 699–713. We run `Structure` Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. “Inference of population structure using multilocus genotype data”. In: *Genetics* 155.2 (2000), pp. 945–959 on the full set of loci for 20 runs for each K from 17 to 21, then align all the 100 runs using `Clumppling`. This dataset is used to demonstrate the alignment of `Structure` output files, using custom color map for visualizations.

Commands:

```
# !/bin/bash

# run with structure output files, all visualizations, and
↪  custom color map
python -m clumppling \
-i examples/chicken \
-o examples/chicken_output \
-f structure \
--extension _f \
--plot_type all \
--fig_format png \
--custom_cmap examples/custom_colors.txt
```

Outputs:

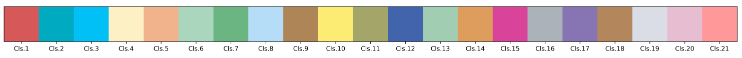


Figure 9: Chicken data: color map.

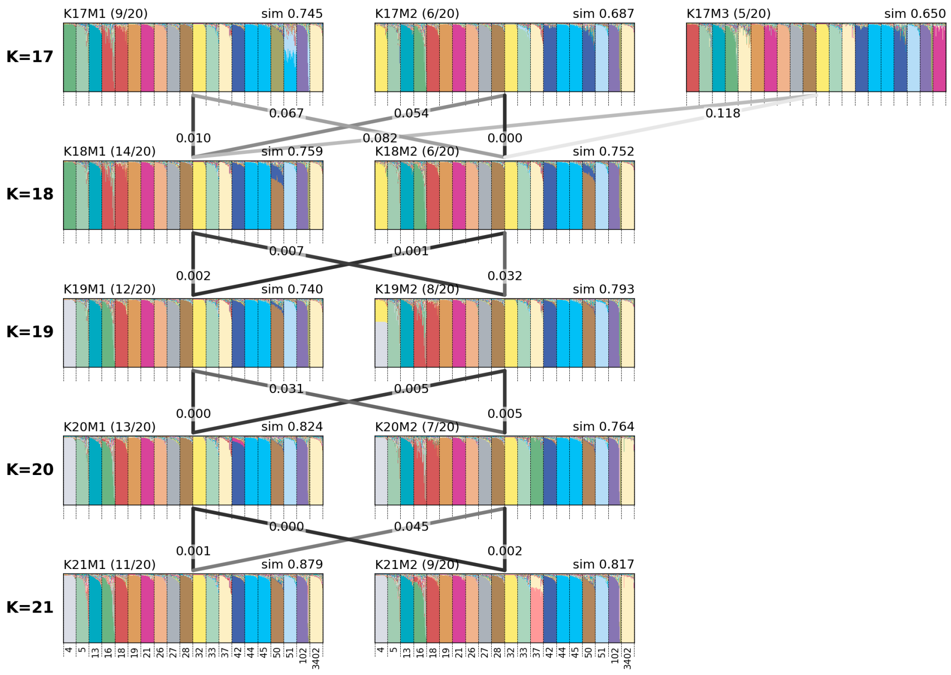


Figure 10: Chicken data: aligned modes.

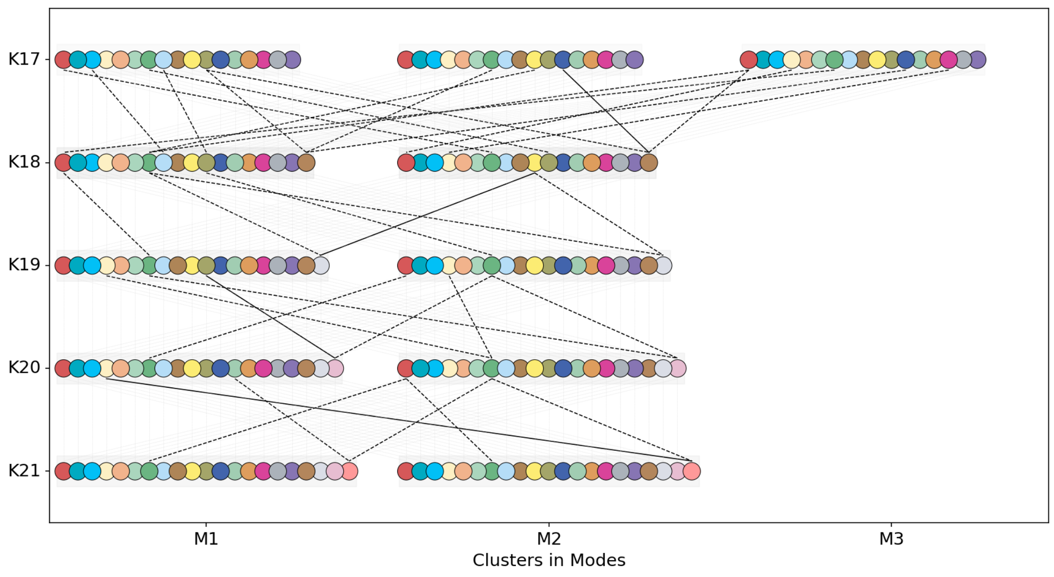


Figure 11: Chicken data: alignment pattern.

7.3 1000 Genome Project data (`Admixture`)

Clustering results of 1000 Genome Project data comprise 8 `Admixture` clustering runs for each *K* from 2 to 8, a total of 56 runs, generated on 2,426 individuals from 26 populations from the 1000 Genome Project data (1000 Genomes Project Consortium et al. “A global reference for human genetic variation”. In: *Nature* 526.7571 (2015), p. 68). These clustering runs were also used in the demonstration of `Pong`Aaron A Behr et al. “pong: Fast analysis and visualization of latent clusters in population genetic data”. In: *Bioinformatics* 32.18 (2016), pp. 2817–2823).

Commands (with algorithmic parameters to reproduce `Pong`’s alignment results and alternating line colors in visualizations to better distinguish between modes):

```
# 1kG from Pong
input_dir=examples/1kG-p3_clumppling_input
pop_label_file=$input_dir/population_labels.txt
output_dir=examples/1kG-p3_clumppling_output
mkdir -p $output_dir

# reproduce Pong's results
```

```
python -m clumppling -i $input_dir -o $output_dir \
-f admixture --extension .Q --ind_labels $pop_label_file \
--test_comm F --cd_res 1.05 --fig_format png \
--plot_type all --regroup_ind T --alt_color T
```

Outputs:

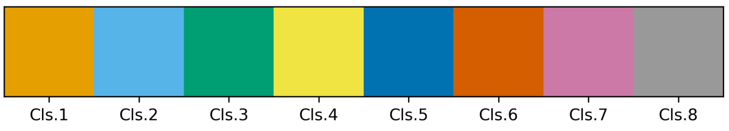


Figure 12: 1kG data: color map.

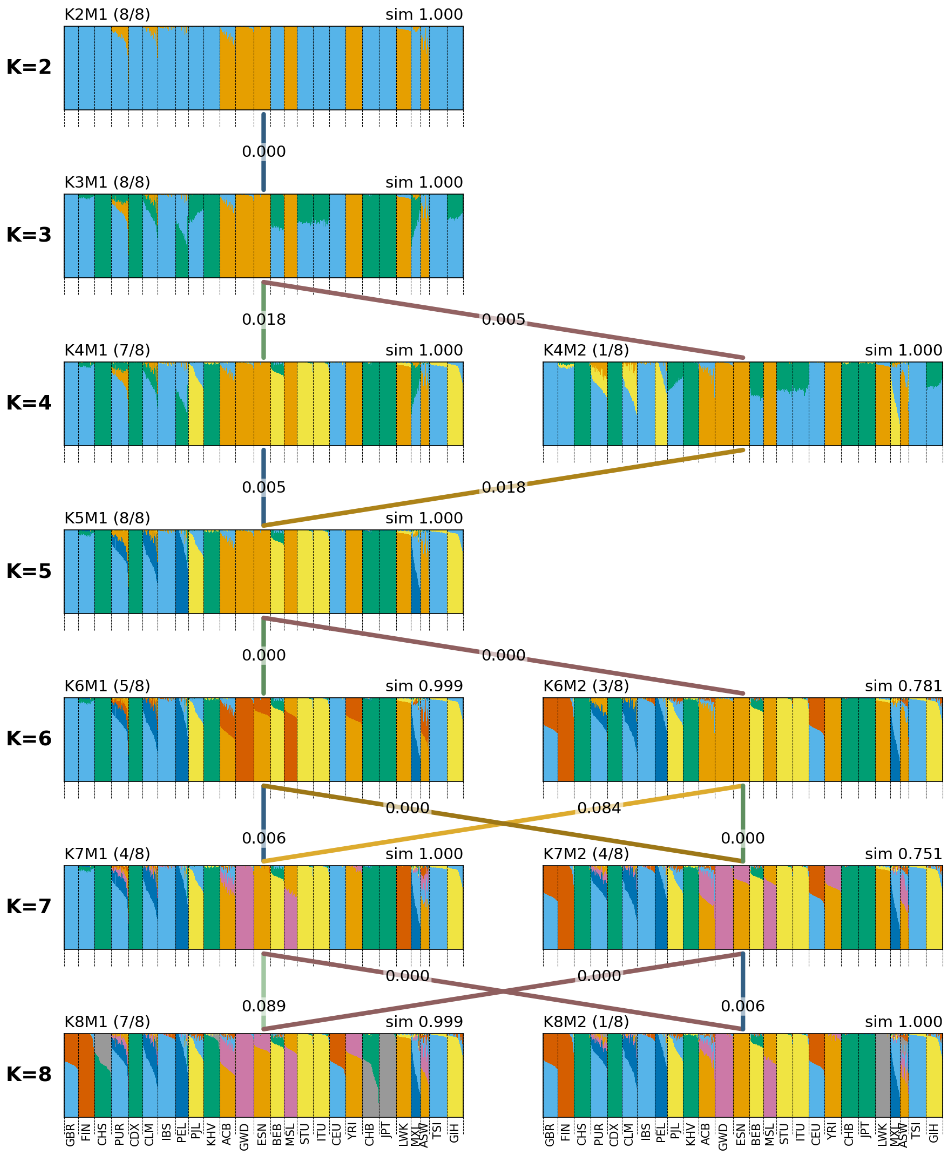


Figure 13: 1kG data: aligned modes.

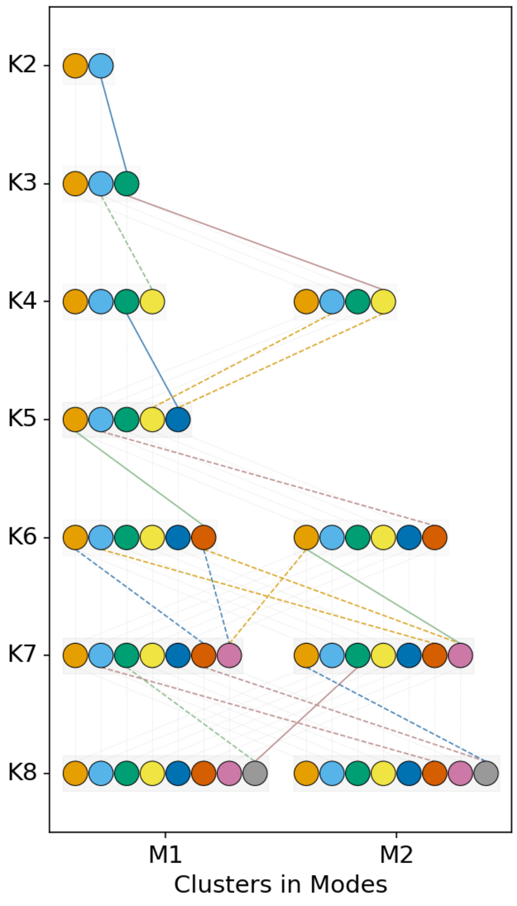


Figure 14: 1kG data: alignment pattern.

8 Submodules

There are five submodules available in `Clumppling`:

- `parseInput` (main workflow): input parsing
- `alignWithinK` (main workflow): alignment of runs with same *K*
- `detectMode` (main workflow): mode detection
- `alignAcrossK` (main workflow): alignment of runs across different *K*
- `compModels` (additional functionality): comparison (and alignment) or results from different models

All submodules can be called by `python -m clumppling.SUBMODULE` followed by appropriate arguments.

8.1 parseInput

The submodule handles reading and parsing of input files that contain the clustering results. It supports various formats and prepares data for downstream analysis.

Usage:

```
usage: __main__.py [-h] -i INPUT -o OUTPUT -f
 ↪  {generalQ,admixture,structure,fastStructure} [--extension
 ↪  EXTENSION] [--skip_rows SKIP_ROWS]
                   [--remove_missing REMOVE_MISSING]

clumppling.parseInput

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file path
  -o OUTPUT, --output OUTPUT
                        Output file directory
  -f {generalQ,admixture,structure,fastStructure}, --format
   ↪  {generalQ,admixture,structure,fastStructure}
                        File format
  --extension EXTENSION
                        Extension of input files
  --skip_rows SKIP_ROWS
                        Skip top rows in input files
  --remove_missing REMOVE_MISSING
                        Remove individuals with missing data:
                         ↪  True/False
```

Arguments:

Table 3: Arguments for `clumppling.parseInput`.

Argument	Description	Data Type
Required Arguments		
`-i, –input`	Input file path.	str (Required)
`-o, –output`	Output file directory.	str (Required)
`-f, –format`	File format. Choices are: `generalQ`, `admixture`, `structure`, `fastStructure`.	str (Required)
Optional Arguments		
`–extension`	Extension of input files.	str (default: " ")
`–skip_rows`	Skip top N rows in input files.	int (default: 0)
`–remove_missing`	Remove individuals with missing data.	bool (default: True)

Example: (reading all files in the provided input path, not skipping rows, removing individuals with missing data):

```
data
python -m clumppling.parseInput \
-i examples/submodules/input \
-o examples/submodules/output \
-f generalQ
```

8.2 `alignWithinK`

This submodule aligns clustering runs within a single value of K.

Usage:

```
usage: __main__.py [-h] [--qfiles [QFILES ...]] [--qfilelist
 ↪  QFILELIST] -o OUTPUT

clumppling.alignWithinK

options:
  -h, --help            show this help message and exit
  --qfiles [QFILES ...]
                        List of Q files to align, passed as
                         ↪  command-line arguments
  --qfilelist QFILELIST
                        A plain text file containing Q file
                        ↪  names (one per line).
  -o OUTPUT, --output OUTPUT
                        Output file name
```

Arguments:

Table 4: Arguments for `clumppling.alignWithinK`.

Argument	Description	Data Type
`−qfiles`	List of Q files to align, passed as one or more command-line arguments.	list[str] (default: None)
`−qfilelist`	A plain text file containing Q file names, one per line.	str (default: None)
`-o, −output`	Output file name for the aligned results.	str (Required)

Example:

```
python -m clumppling.alignWithinK \
--qfilelist examples/submodules/K3.qfilelist \
-o examples/submodules/K3_aligned.txt

python -m clumppling.alignWithinK \
--qfilelist examples/submodules/K5.qfilelist \
-o examples/submodules/K5_aligned.txt
```

8.3 `detectMode`

This submodule detects modes (distinct clustering solutions) among multiple clustering runs for a given *K* after within-*K* alignment.

Usage:

```
usage: __main__.py [-h] --align_res ALIGN_RES --qfilelist
↪  QFILELIST -o OUTPUT [--qnamelist QNAMELIST]
                     [--cd_method {louvain,leiden,infomap,markov_⌋
                      ↪  clustering,label_propagation,walktrap,⌋
                      ↪  custom}] [--cd_res CD_RES] [--test_comm
                      ↪  TEST_COMM]
```

```
                   [--comm_min COMM_MIN] [--comm_max COMM_MAX]

clumppling.alignWithinK

options:
  -h, --help            show this help message and exit
  --align_res ALIGN_RES
                        Path to the alignment results file
  --qfilelist QFILELIST
                        A plain text file containing Q file
                         ↪  names (one per line).
  -o OUTPUT, --output OUTPUT
                        Output file directory
  --qnamelist QNAMELIST
                        A plain text file containing replicate
                         ↪  names (one per line) (default: file
                         ↪  base from qfilelist)
  --cd_method {louvain,leiden,infomap,markov_clustering,label_
   ↪  propagation,walktrap,custom}
                        Community detection method to use
                         ↪  (default: louvain)
  --cd_res CD_RES       Resolution parameter for the default
   ↪  Louvain community detection (default: 1.0)
  --test_comm TEST_COMM
                        Whether to test community structure
                         ↪  (default: True)
  --comm_min COMM_MIN   Minimum threshold for cost matrix
   ↪  (default: 1e-4)
  --comm_max COMM_MAX   Maximum threshold for cost matrix
   ↪  (default: 1e-2)
```

Arguments:

Table 5: Arguments for `clumppling.detectMode`.

Argument	Description	Data Type
Required Arguments		
`–align_res`	Path to the alignment results file.	str (Required)

Continued on next page

Table 5: Arguments for `clumppling.detectMode` (continued).

Argument	Description	Data Type
`−qfilelist`	A plain text file containing Q file names, one per line.	str (Required)
`-o, −output`	Output file directory.	str (Required)
Optional Arguments		
`−qnamelist`	A plain text file containing replicate names (one per line).	str (default: '[]')
`−cd_method`	Community detection method to use. Choices include: `louvain`, `leiden`, `infomap`, etc.	str (default: 'louvain')
`−cd_res`	Resolution parameter for the Louvain community detection.	float (default: 1.0)
`−test_comm`	Whether to test community structure.	bool (default: True)
`−comm_min`	Minimum threshold for cost matrix.	float (default: 1e-4)
`−comm_max`	Maximum threshold for cost matrix.	float (default: 1e-2)

Example:

```
python -m clumppling.detectMode \
--align_res examples/submodules/K3_aligned.txt \
--qfilelist examples/submodules/K3.qfilelist \
--qnamelist examples/submodules/K3.qnamelist \
--cd_method markov_clustering \
-o examples/submodules/K3_modes

python -m clumppling.detectMode \
--align_res examples/submodules/K5_aligned.txt \
--qfilelist examples/submodules/K5.qfilelist \
--qnamelist examples/submodules/K5.qnamelist \
--cd_method markov_clustering \
-o examples/submodules/K5_modes
```

8.4 `alignAcrossK`

This submodule aligns clustering runs (modes) different values of K.

Usage:

```
usage: __main__.py [-h] [--qfilelist QFILELIST] -o OUTPUT
 ↪  [--qnamelist QNAMELIST] [--use_best_pair USE_BEST_PAIR]

clumppling.alignAcrossK

options:
  -h, --help            show this help message and exit
  --qfilelist QFILELIST
                        A plain text file containing Q file
                         ↪  names (one per line).
  -o OUTPUT, --output OUTPUT
                        Directory to save output files
  --qnamelist QNAMELIST
                        A plain text file containing replicate
                         ↪  names (one per line) (default: file
                         ↪  base from qfilelist)
  --use_best_pair USE_BEST_PAIR
                        Use best pair as anchor for across-K
                         ↪  alignment (alternative: major):
                         ↪  True (default)/False
```

Arguments:

Table 6: Arguments for `clumppling.alignAcrossK`.

Argument	Description	Data Type
`–qfilelist`	A plain text file containing Q file names, one per line.	str (default: None)
`-o, –output`	Directory to save output files.	str (Required)
`–qnamelist`	A plain text file containing replicate names (one per line).	str (default: '[]')
`–use_best_pair`	Use the best pair as an anchor for alignment (alternative: major).	bool (default: True)

Example:

```
# prepare mode files
for K in 3 5; do
    SRC=examples/submodules/K${K}_modes
    DST=examples/submodules/K3K5_modes
    mkdir -p $DST
    for f in "$SRC"/*.Q; do
        if [ -f "$f" ]; then
            cp "$f" "$DST/K${K}$(basename "$f")"
        fi
    done
done
# generate list of mode files and names
ls examples/submodules/K3K5_modes/*_rep.Q >
↪   examples/submodules/K3K5_modes/K3K5_modes.qfilelist
for f in examples/submodules/K3K5_modes/*_rep.Q; do [ -f "$f" ]
↪   && basename "$f" | sed 's/\_rep.Q$//' >>
↪   examples/submodules/K3K5_modes/K3K5_modes.qnamelist; done

# run clumppling
python -m clumppling.alignAcrossK \
--qfilelist
↪   examples/submodules/K3K5_modes/K3K5_modes.qfilelist \
--qnamelist
↪   examples/submodules/K3K5_modes/K3K5_modes.qnamelist \
-o examples/submodules/K3K5_acrossK_output
```

8.5 compModels

This submodule compares clustering results from different models, with potentially different *K* values for the results of each model, and displays the aligned results side by side.

Usage:

```
usage: __main__.py [-h] --models MODELS [MODELS ...]
↪   --qfilelists QFILELISTS [QFILELISTS ...] [--qnamelists
↪   QNAMELISTS [QNAMELISTS ...]]
                   [--mode_stats_files MODE_STATS_FILES
                   ↪   [MODE_STATS_FILES ...]] [--ind_labels
                   ↪   IND_LABELS] -o OUTPUT [-v VIS]
                   ↪   [--custom_cmap CUSTOM_CMAP]
                   [--include_sim_in_label INCLUDE_SIM_IN_LABEL]
                     ↪  [--fig_format {png,jpg,jpeg,tif,tiff,s⌋
                     ↪  vg,pdf,eps,ps,bmp,gif}]

clumppling.compModels

options:
  -h, --help            show this help message and exit
  --models MODELS [MODELS ...]
                        List of model names.
  --qfilelists QFILELISTS [QFILELISTS ...]
                        List of files containing Q file names
                         ↪  from each model.
  --qnamelists QNAMELISTS [QNAMELISTS ...]
                       List of files containing replicate names
                         ↪  from each model.
  --mode_stats_files MODE_STATS_FILES [MODE_STATS_FILES ...]
                       List of files containing mode statistics
                         ↪  from each model.
  --ind_labels IND_LABELS
                        A plain text file containing individual
                         ↪  labels (one per line)
  -o OUTPUT, --output OUTPUT
                        Output file directory
  -v VIS, --vis VIS     Whether to generate figure(s): True
   ↪  (default)/False
  --custom_cmap CUSTOM_CMAP
                        A plain text file containing customized
                         ↪  colors (one per line; in hex code):
                         ↪  if empty (default), using the
                         ↪  default colormap, otherwise use the
                         ↪  user-
                        specified colormap
  --include_sim_in_label INCLUDE_SIM_IN_LABEL
                        Whether to include (original) alignment
                         ↪  similarity in mode labels (if
                         ↪  provided): True (default)/False
  --fig_format {png,jpg,jpeg,tif,tiff,svg,pdf,eps,ps,bmp,gif}
                       Figure format for output files (default:
                         ↪  tiff)
```

Arguments:

Table 7: Arguments for `clumppling.compModels`.

Argument	Description	Data Type
Required Arguments		
`–models`	List of model names.	list[str] (Required)
`–qfilelists`	List of files containing Q file names from each model.	list[str] (Required)
`-o, –output`	Output file directory.	str (Required)
Optional Arguments		
`–qnamelists`	List of files containing replicate names from each model.	list[str] (default: "")
`–mode_stats_files`	List of files containing mode statistics from each model.	list[str] (default: "")
`-v, –vis`	Whether to generate figure(s).	bool (default: True)
`–custom_cmap`	A plain text file containing customized colors (one per line; in hex code). If empty, a default colormap is used.	str (default: "")

Example:

```
# prepare files
$comp_dir=PATH_TO_RES_FOLDER
mkdir -p $comp_dir

model1_dir=PATH_TO_MODEL1_CLUMPPLING_OUTPUT
model2_dir=PATH_TO_MODEL2_CLUMPPLING_OUTPUT

# should not contain underscore or dash in the model name
model1=MODEL1
model1_suffix=avg
model2=MODEL2
model2_suffix=avg

ls $model1_dir/modes_aligned/*_${model1_suffix}.Q >
 ↪  $comp_dir/${model1}.qfilelist
for f in $model1_dir/modes_aligned/*_${model1_suffix}.Q; do [
 ↪  -f "$f" ] && basename "$f" | sed
 ↪  "s/\_${model1_suffix}.Q$//" >>
 ↪  $comp_dir/${model1}.qnamelist; done
```

```
ls $model2_dir/modes_aligned/*_${model2_suffix}.Q >
 ↪  $comp_dir/${model2}.qfilelist
for f in $model2_dir/modes_aligned/*_${model2_suffix}.Q; do [
 ↪  -f "$f" ] && basename "$f" | sed
 ↪  "s/\_${model2_suffix}.Q$//" >>
 ↪  $comp_dir/${model2}.qnamelist; done
cp $model1_dir/modes/mode_stats.txt
 ↪  $comp_dir/${model1}_mode_stats.txt
cp $model2_dir/modes/mode_stats.txt
 ↪  $comp_dir/${model2}_mode_stats.txt
cp $model1_dir/input/ind_labels_grouped.txt
 ↪  $comp_dir/ind_labels_grouped.txt

python -m clumppling.compModels \
--models ${model1} ${model2} \
--qfilelists $comp_dir/${model1}.qfilelist
 ↪  $comp_dir/${model2}.qfilelist \
--qnamelists $comp_dir/${model1}.qnamelist
 ↪  $comp_dir/${model2}.qnamelist \
--mode_stats_files $comp_dir/${model1}_mode_stats.txt
 ↪  $comp_dir/${model2}_mode_stats.txt \
--ind_labels $comp_dir/ind_labels_grouped.txt \
--fig_format png \
--output $comp_dir/comp_${model1}_vs_${model2}
```

Output:

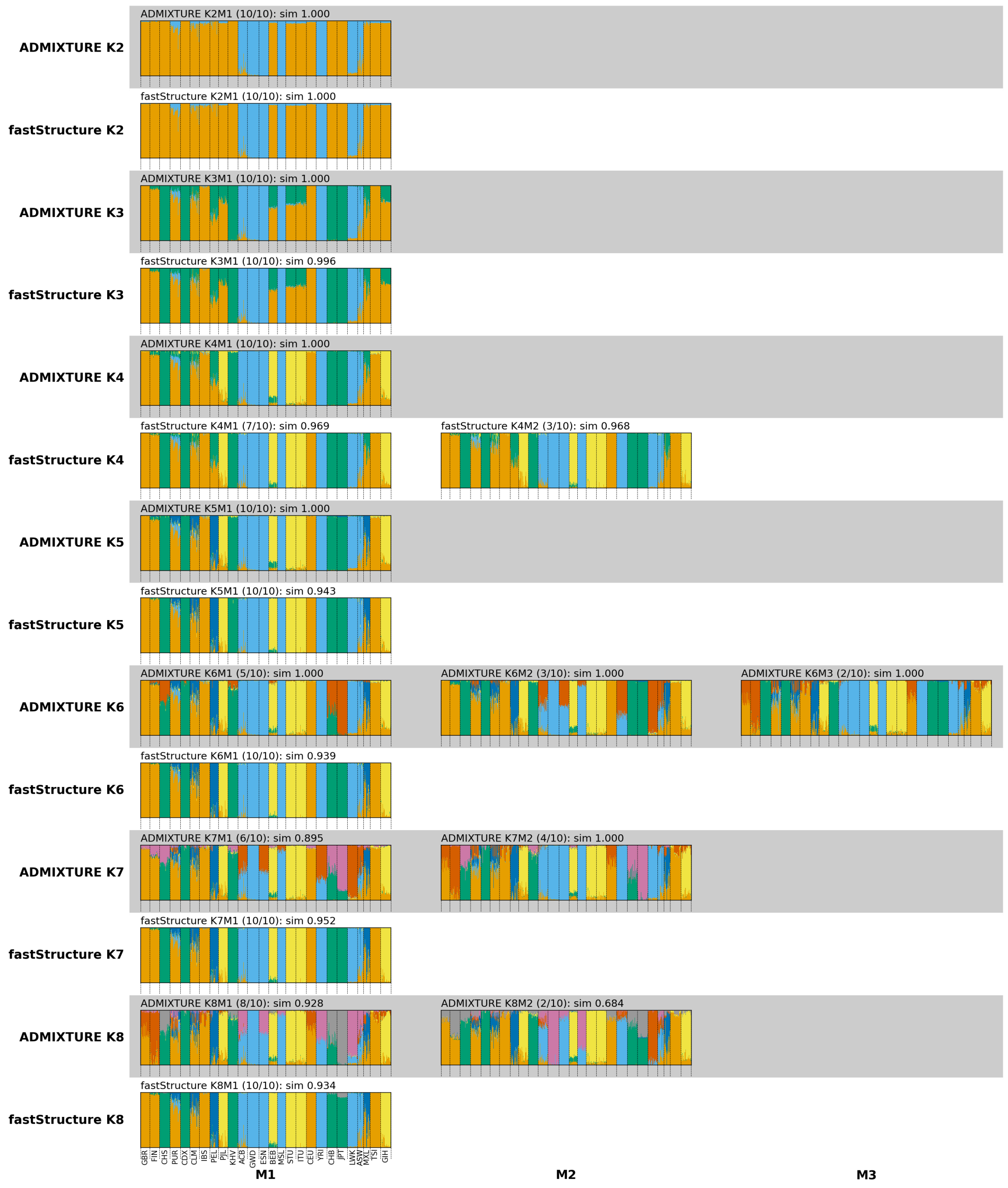


Figure 15: Model comparison output: aligned modes in (interleaving) graph layout. Modes from different models are grouped by *K* and organized in rows with alternating background colors.

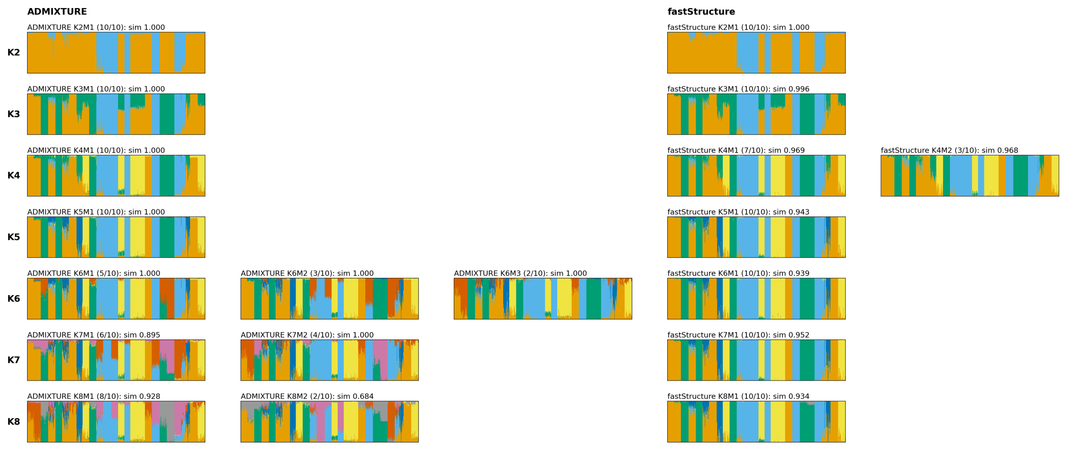


Figure 16: Model comparison output: aligned modes in (side-by-side) graph layout. Modes from different models are put side by side.

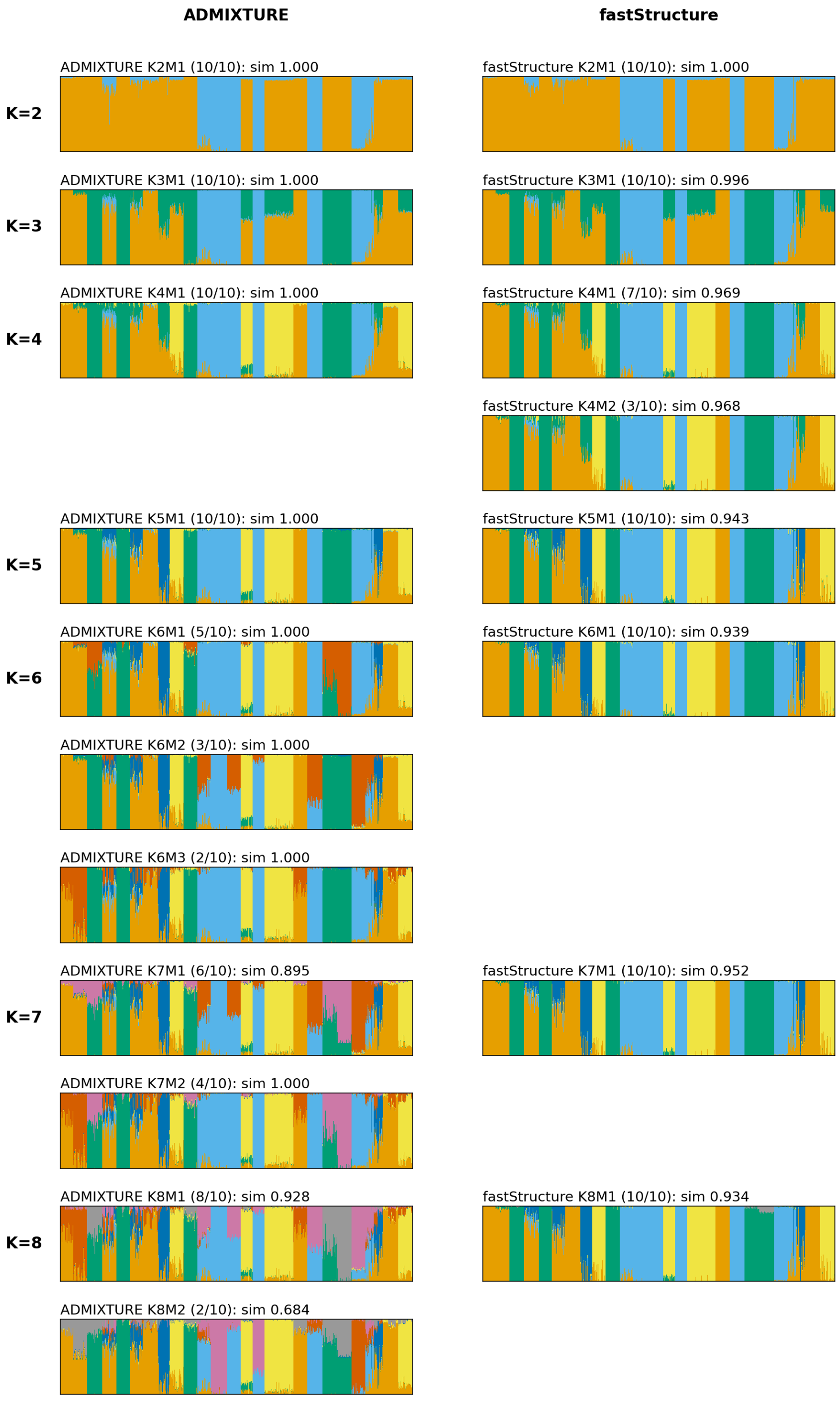


Figure 17: Model comparison output: aligned modes in list layout. Modes from different models are each present in a list of structure plots, side by side.

8.6 Generation of figures

The following code snippets provide examples for generating and customizing figures.

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from clumppling.plot import parse_custom_cmap, plot_colorbar,
 ↪  plot_alignment_list, plot_alignment_graph,
 ↪  plot_memberships_list, plot_graph
from clumppling.utils import str_to_pattern

K_max = 5
K_range = [3,5]
cmap_file = "examples/custom_colors.txt"
acrossK_dir = "examples/submodules/K3K5_acrossK_output"
mode_dir = os.path.join(acrossK_dir, "modes_aligned")
ind_label_file = "examples/capeverde_ind_labels.txt"
fig_dir = "examples/submodules/figures"
os.makedirs(fig_dir, exist_ok=True)

# Read the individual labels
ind_labels = []
with open(ind_label_file, "r") as f:
    for line in f:
        ind_labels.append(line.strip())
print(f"Loaded {len(ind_labels)} individual labels from
 ↪  {ind_label_file}")

# Read the custom color map from the file
with open(cmap_file, "r") as f:
    colors = f.read().strip().splitlines()
cmap = parse_custom_cmap(colors, K=K_max)
# Plot the colorbar
plot_colorbar(cmap,K_max,fig_dir)

# Load membership matrices
if not os.path.exists(mode_dir):
    raise FileNotFoundError(f"Mode directory {mode_dir} does
     ↪  not exist.")
```

```
Q_list = []
mode_names = []
for f in os.listdir(mode_dir):
    if f.endswith(".Q"):
        suffix = f.strip(".Q")
        Q_list.append(np.loadtxt(os.path.join(mode_dir, f)))
        mode_names.append(suffix)
mode_K = [Q.shape[1] for Q in Q_list]
assert mode_K == sorted(mode_K), "Modes should be sorted by K."

Q_list_list = []
mode_names_list = []
for K in K_range:
    Q_list_list.append([Q for i, Q in enumerate(Q_list) if
     ↪  mode_K[i] == K])
    mode_names_list.append([name for i, name in
     ↪  enumerate(mode_names) if mode_K[i] == K])

# Load alignment results
all_modes_alignment = {}
with open(os.path.join(mode_dir, "all_modes_alignment.txt"),
 ↪  "r") as f:
    for line in f:
        parts = line.strip().split(":")
        if len(parts) < 2:
            continue
        mode_name = parts[0]
        alignment = str_to_pattern(parts[1])
        all_modes_alignment[mode_name] = alignment

alignment_acrossK = {}
cost_acrossK = {}
with open(os.path.join(acrossK_dir, "alignment_acrossK.txt"),
 ↪  "r") as f:
    next(f)
    for line in f:
        parts = line.strip().split(",")
        if len(parts) < 3:
            continue
        mode_pair = parts[0]
        alignment = str_to_pattern(parts[2])
        alignment_acrossK[mode_pair] = alignment
        cost = float(parts[1])
        cost_acrossK[mode_pair] = cost

# Plot alignment pattern (as a list)
fig = plot_alignment_list(mode_K, mode_names, cmap,
 ↪  alignment_acrossK, all_modes_alignment, marker_size=200)
fig.savefig(os.path.join(fig_dir,"alignment_pattern_list_{}.p⌋
 ↪  ng".format(suffix)), bbox_inches='tight', dpi=150,
 ↪  transparent=False)
plt.close(fig)


fig = plot_alignment_graph(K_range, names_list=mode_names_list,
 ↪  cmap=cmap,
                           alignment_acrossK=alignment_acrossK,
                            ↪  all_modes_alignment=all_modes_⌋
                            ↪  alignment,
                           ls_alt=['-', '--'],
                            ↪  color_alt=['dimgrey',
                            ↪  'darkslateblue','darkgreen'])
fig.savefig(os.path.join(fig_dir,"alignment_pattern_graph_{}.⌋
 ↪  png".format(suffix)), bbox_inches='tight', dpi=150,
 ↪  transparent=False)
plt.close(fig)


# Plot aligned memberships (as a list, with individuals ordered
 ↪  by major cluster's memberships, based on major mode of
 ↪  smallest K)
Q_ref = Q_list_list[0][0] # use the major mode of smallest K as
 ↪  reference
fig = plot_memberships_list(Q_list, cmap, names=mode_names,
 ↪  ind_labels=ind_labels,
                            order_refQ=Q_ref,
                             ↪  order_cls_by_label=True)
fig.tight_layout()
fig.savefig(os.path.join(fig_dir,
 ↪  "aligned_membership_list.png"), bbox_inches='tight',
 ↪  dpi=150, transparent=False)
```

```
plt.close(fig)

# Plot aligned memberships (as a graph, with individuals ordered
↪  by major cluster's memberships, based on major mode of
↪  largest K)
Q_ref = Q_list_list[-1][0] # use the major mode of largest K as
↪  reference
fig = plot_graph(K_range, Q_list_list, cmap,
                 names_list=mode_names_list, labels_list=None,
                 cost_acrossK=cost_acrossK,
                 ↪  ind_labels=ind_labels,
                 fontsize=14, line_cmap=plt.get_cmap("Greys"),
                 order_refQ=Q_ref, order_cls_by_label=True)
fig.savefig(os.path.join(fig_dir,
↪  "aligned_membership_graph.png"), bbox_inches='tight',
↪  dpi=150, transparent=False)
plt.close(fig)

# Plot aligned memberships (as a graph, with individuals
↪  unordered)
fig = plot_graph(K_range, Q_list_list, cmap,
                 names_list=mode_names_list, labels_list=None,
                 cost_acrossK=cost_acrossK,
                 ↪  ind_labels=ind_labels,
                 fontsize=14, line_cmap=plt.get_cmap("Greys"),
                 order_refQ=None, order_cls_by_label=False)
fig.savefig(os.path.join(fig_dir,
↪  "aligned_membership_graph_unordered.png"),
↪  bbox_inches='tight', dpi=150, transparent=False)
plt.close(fig)
```

Acknowledgements

This manual was prepared using the Technical Document Template from Overleaf[18], with assistance from Google's Gemini 2.5 Pro for refining phrasing and formatting.

To report bugs, ask questions, or provide feedback, please open a GitHub issue or email the author directly[19].

If you use this software in your research, please cite the following publication:

> Liu, X., Kopelman, N. M., & Rosenberg, N. A. (2024). Clumppling: cluster matching and permutation program with integer linear programming. *Bioinformatics, 40*(1), btad751.

[18]`https://www.overleaf.com/latex/templates/technical-document-template/mdgftpdfbvbs`

[19]xiranliu [at] stanford [dot] edu