pyGenClean - Automated
Data Clean Up

pyGenClean Documentation

Release 1.8.3

Louis-Philippe Lemieux Perreault

Mar 09, 2017

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

ONE

INTRODUCTION

Genetic association studies making use of high throughput genotyping arrays need to process large amounts of data in the order of millions of markers per experiment. The first step of any analysis with genotyping arrays is typically the conduct of a thorough data clean up and quality control to remove poor quality genotypes and generate metrics to inform and select individuals for downstream statistical analysis.

pyGenClean is a bioinformatics tool to facilitate and standardize the genetic data clean up pipeline with genotyping array data. In conjunction with a source batch-queuing system, the tool minimizes data manipulation errors, it accelerates the completion of the data clean up process and it provides informative graphics and metrics to guide decision making for statistical analysis.

pyGenClean is a command tool working on both Linux and Windows operating systems. Its usage is shown below:

```
$ run_pyGenClean --help
usage: run_pyGenClean [-h] [-v] [--bfile FILE] [--tfile FILE] [--file FILE]
                      [--report-title TITLE] [--report-author AUTHOR]
                      [--report-number NUMBER]
                      [--report-background BACKGROUND] --conf FILE

Runs the data clean up (pyGenClean version 1.8.3).

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively).
  --tfile FILE          The input file prefix (will find the plink transposed
                        files by appending the prefix to the .tped and .tfam
                        files, respectively).
  --file FILE           The input file prefix (will find the plink files by
                        appending the prefix to the .ped and .fam files).

Report Options:
  --report-title TITLE  The report title. [default: Genetic Data Clean Up]
  --report-author AUTHOR
                        The current project number. [default: pyGenClean]
  --report-number NUMBER
                        The current project author. [default: Simple Project]
  --report-background BACKGROUND
                        Text of file containing the background section of the
                        report.
```

```
Configuration File:
  --conf FILE           The parameter file for the data clean up.
```

The tool consists of multiple standalone scripts that are linked together via a main script (`run_pyGenClean`) and a configuration file (the `--conf` option), the latter facilitating user customization.

The *Data clean up protocol schema* shows the proposed data cleanup pipeline. Each box represents a customizable standalone script with a quick description of its function. Optional manual checks for go-no-go decisions are indicated.

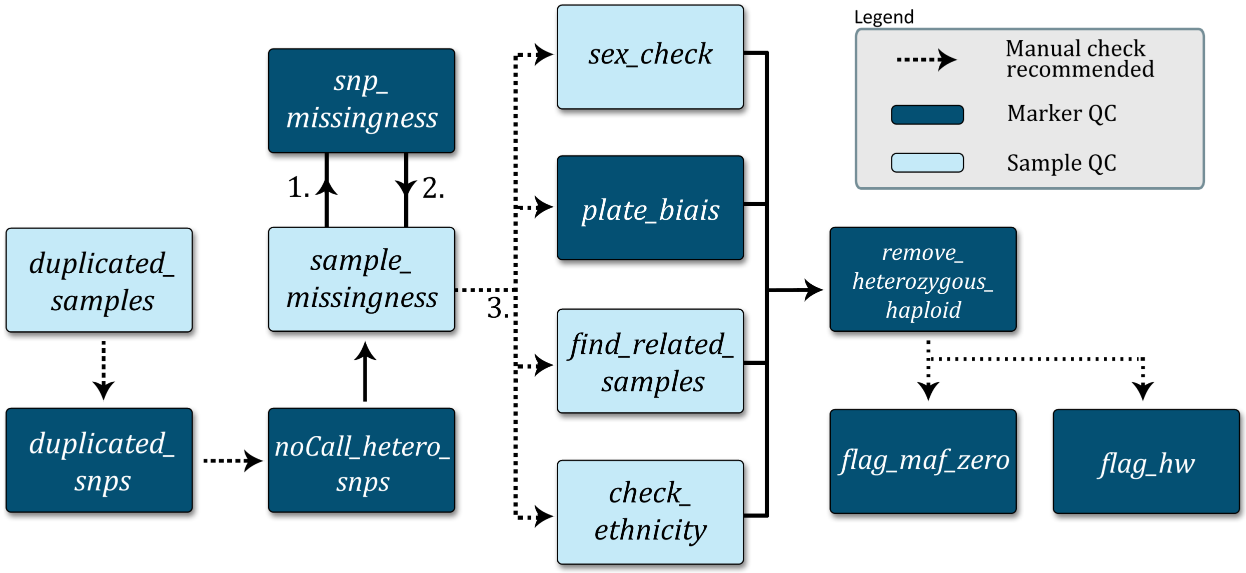

Fig. 1.1: Data clean up protocol schema

CHAPTER

TWO

INSTALLATION

pyGenClean is a Python package that works on both Linux and Windows operating systems. It requires a set of Python dependencies and PLINK. Complete installation procedures are available for both Linux (32 and 64 bits) and Windows in the following sections.

2.1 Linux Installation

The following steps will help you install pyGenClean on a Linux machine.

2.1.1 Requirements

The following softwares and packages are required for pyGenClean:

1. Python 2.7
2. PLINK (1.07)
3. `numpy` (version 1.6.2 or latest)
4. `matplotlib` (version 1.2.0 or latest)
5. `scipy` (version 0.11.0 or latest)
6. `scikit-learn` (version 0.12.1 or latest)
7. `Jinja2` (version 2.7.3 or latest)

Note: All the requirements will be installed along with the main pyGenClean module.

Warning: The *Plink* software needs to be in the *PATH* (or in the current working directory). In other words, you should be able to type *plink* at the command line.

2.1.2 Installation

There are two main ways to to install pyGenClean: using an existing Python 2.7 distribution and creating a Python *Virtual environment*, or using *Miniconda*.

Virtual environment

If python is already installed, a python virtual environment should be created. If one is already present, you can just proceed to the *Activating the environment* section.

To create a new python virtual environment, download the latest version of `virtualenv`, located at this web page: http://pypi.python.org/pypi/virtualenv. At the moment of writing this documentation, the latest version was 13.1.2, and the file was named `virtualenv-13.1.2.tar.gz`. Locate the archive, which is usually in the `~/Downloads` directory.

```
$ cd ~/Downloads
$ tar -zxf virtualenv-13.1.2.tar.gz
$ cd virtualenv-13.1.2
```

There is no need to install the module. Just create a directory and create the Python virtual environment:

```
$ mkdir -p ~/softwares/Python-2.7_virtualenv
$ python ./virtualenv.py \
>     --no-site-packages \
>     ~/softwares/Python-2.7_virtualenv
```

Activating the environment

To activate the Python virtual environment, perform the following command:

```
$ source ~/softwares/Python-2.7_virtualenv/bin/activate
```

Finally, to deactivate the Python virtual environment, either close the terminal, or perform the following command:

```
$ deactivate
```

> **Warning:** For the following installations and tests, be certain that the Python virtual environment is activated, or nothing will work as planned...
>
> The best way to know if the Python virtual environment is activated, is to see its name, in parenthesis, before the usual prompt in the terminal. For example:
>
> ```
> (Python-2.7_virtualenv)[username@localhost ~]$
> ```

Installing pyGenClean

To install *`pyGenClean`*, only perform the following command:

```
$ pip install pyGenClean
```

Updating pyGenClean

To update *`pyGenClean`*, perform the following command:

```
$ pip install -U pyGenClean
```

Miniconda

Download and install miniconda (located at http://conda.pydata.org/miniconda.html). By default, miniconda is installed in `~/miniconda`.

To create a new virtual environment, perform the following command:

```
$ conda create -n Python-2.7_virtualenv python=2
```

Activating the conda environment

To activate the Python virtual environment (miniconda), perform the following command:

```
$ source ~/miniconda/bin/activate Python-2.7_virtualenv
```

Finally, to deactivate the Python virtual environment, either close the terminal, or perform the following command:

```
$ source deactivate
```

> **Warning:** For the following installations and tests, be certain that the Python virtual environment is activated, or nothing will work as planned...
>
> The best way to know if the Python virtual environment is activated, is to see its name, in parenthesis, before the usual prompt in the terminal. For example:
>
> ```
> (Python-2.7_virtualenv)[username@localhost ~]$
> ```

Installing pyGenClean

To install *pyGenClean*, only perform the following command:

```
$ conda install pyGenClean -c http://statgen.org/wp-content/uploads/Softwares/
↪pyGenClean
```

Updating pyGenClean

To update *pyGenClean*, perform the following command:

```
$ conda update pyGenClean -c http://statgen.org/wp-content/uploads/Softwares/
↪pyGenClean
```

2.1.3 Testing the installation

> **Warning:** Before using *pyGenClean*, be certain that the previously installed Python virtual environment is activated (see *Activating the environment* or *Activating the conda environment* for more information). If the proper environment is not activated, noting will work...

To test the algorithm, download the test data from http://statgen.org/downloads/pygenclean/ and the HapMap reference populations (build 37).

Locate the downloaded archives (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ mkdir -p ~/test_pyGenClean
$ tar -C ~/test_pyGenClean -jxf check_ethnicity_HapMap_reference_populations_b37.tar.
↪bz2
$ tar -C ~/test_pyGenClean -jxf pyGenClean_test_data.tar.bz2
$ cd ~/test_pyGenClean
```

Create a text file named `conf.ini` inside the `~/test_pyGenClean` directory, containing the following text:

```
[1]
script = check_ethnicity
ceu-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_CEU_r23a_filtered_b37
yri-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_YRI_r23a_filtered_b37
jpt-chb-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_JPT_CHB_r23a_filtered_b37
nb-components = 2
multiplier = 1

[2]
script = sex_check
```

Run the following command:

```
$ run_pyGenClean \
>     --conf conf.ini \
>     --bfile pyGenClean_test_data/1000G_EUR-MXL_Human610-Quad-v1_H
```

Results

Valuable information will be shown in the terminal. Once the program has finished, the results are in the new directory `data_clean_up.date_time` where `date` is the current date, and `time` is the time at which the analysis started.

Here are the new directory structure, with only the files you might be interested in:

- data_clean_up.date_time/
  - pyGenClean.log
  - automatic_report.tex
  - excluded_markers.txt
  - excluded_samples.txt
  - 1_check_ethnicity/
    - ethnicity.before.png
    - ethnicity.outliers.png
    - ethnicity.outliers
    - ethnicity.population_file_outliers
  - 2_sex_check/
    - sexcheck.list_problem_sex

The file `pyGenClean.log` contain the information that was displayed in the console. The file `automatic_report.txt` contain the automatic report generated by `pyGenClean`. The files `excluded_markers.txt` and `excluded_samples.txt` contains the list of markers and samples, respectively, that were excluded from the dataset (with the reason).

1_check_ethnicity/

The first image in the first directory (*ethnicity.before.png*) shows the MDS values for each sample before outlier detection. The second image (*ethnicity.outliers.png*) shows the outliers that should be removed for further analysis. Finally, the file `ethnicity.outliers` include a list of samples that should be removed for further analysis. **The total number of outliers for this test should be exactly 63**, but the figures might be mirrored for 32 bits systems. For more information about the results of this module, refer to Section *Ethnicity Module*.

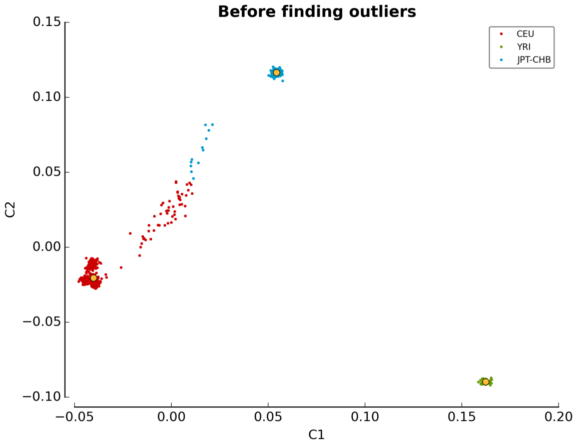

Fig. 2.1: ethnicity.before.png

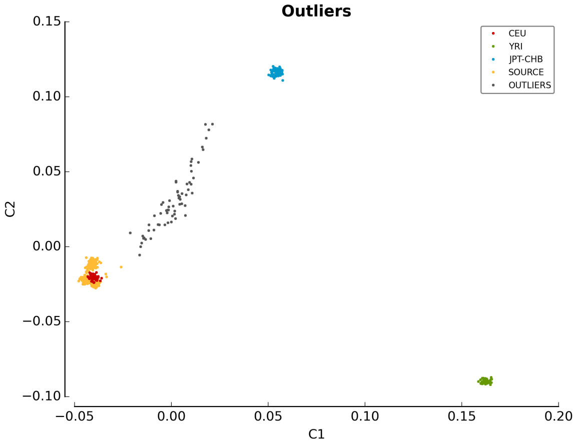

Fig. 2.2: ethnicity.outliers.png

2_sex_check/

In the second directory, there should be a file containing the list of samples with gender problem. **There should be exactly 4 samples with gender problem.** For more information about this module, refer to Section *Sex Check Module*.

If you want to compare your results with the expected ones, just download the files in the archive `pyGenClean_expected_results.tar.bz2`, available through http://statgen.org/downloads/pygenclean/. They were generated using Fedora 18 (64 bits) in about 20 minutes. You should at least compare the following files:

- `1_check_ethnicity`
  - `ethnicity.outliers`
  - `ethnicity.population_file_outliers`
  - All the figures (they might be mirrored).
- `2_sex_check`
  - `sexcheck.list_problem_sex`
  - `sexcheck.list_problem_sex_ids`

Automatic report

If LaTeX is installed, you can perform the following commands to compile the automatic report into a PDF file.

```
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
```

The following PDF report will be generated.

2.2 Windows Installation

The following steps will help you install *pyGenClean* on a Windows machine. It has been tested on both Windows XP and Windows 7.

2.2.1 Requirements

The following softwares and packages are required for `pyGenClean`:

1. Python 2.7
2. PLINK (1.07)
3. `numpy` (version 1.6.2 or latest)
4. `matplotlib` (version 1.2.0 or latest)
5. `scipy` (version 0.11.0 or latest)
6. `scikit-learn` (version 0.12.1 or latest)
7. `Jinja2` (version 2.7.3 or latest)

Note: All the requirements will be installed along with the main *pyGenClean* module.

Warning: The *Plink* software needs to be in the *PATH* (or in the current working directory). In other words, you should be able to type *plink* at the command line.

2.2.2 Installation

The easiest way to install Python on windows is by using *Miniconda*.

Miniconda

Download and install miniconda (located at http://conda.pydata.org/miniconda.html).

To create a new virtual environment, perform the following command:

```
$ conda create -n Python-2.7_virtualenv python=2
```

Activating the conda environment

To activate the Python virtual environment (miniconda), perform the following command:

```
$ activate Python-2.7_virtualenv
```

Installing pyGenClean

To install *pyGenClean*, only perform the following command:

```
$ conda install pyGenClean -c http://statgen.org/wp-content/uploads/Softwares/
↪pyGenClean
```

Updating pyGenClean

To update *pyGenClean*, perform the following command:

```
$ conda update pyGenClean -c http://statgen.org/wp-content/uploads/Softwares/
↪pyGenClean
```

2.2.3 Testing the Algorithm

To test the algorithm, download the test data from http://statgen.org/downloads/pygenclean/ and the HapMap reference populations (build 37). Create a directory on your Desktop named `pyGenClean_test`, and extract the two archive into it. You should have the following directory structure:

```
Desktop\
    pyGenClean_test_data\
        1000G_EUR-MXL_Human610-Quad-v1_H.bed
        1000G_EUR-MXL_Human610-Quad-v1_H.bim
        1000G_EUR-MXL_Human610-Quad-v1_H.fam
    check_ethnicity_HapMap_ref_pops_b37\
        hapmap_CEU_r23a_filtered_b37.bed
        hapmap_CEU_r23a_filtered_b37.bim
        hapmap_CEU_r23a_filtered_b37.fam
        hapmap_YRI_r23a_filtered_b37.bed
        hapmap_YRI_r23a_filtered_b37.bim
        hapmap_YRI_r23a_filtered_b37.fam
        hapmap_JPT_CHB_r23a_filtered_b37.bed
        hapmap_JPT_CHB_r23a_filtered_b37.bim
        hapmap_JPT_CHB_r23a_filtered_b37.fam
```

Open the command prompt and navigate to the newly created directory, and created an new text file using `notepad`:

```
> cd Desktop\pyGenClean_test
> notepad conf.ini
```

Insert the following code in the file:

```
[1]
script = check_ethnicity
ceu-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_CEU_r23a_filtered_b37
yri-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_YRI_r23a_filtered_b37
jpt-chb-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_JPT_CHB_r23a_filtered_b37
nb-components = 2
multiplier = 1

[2]
script = sex_check
```

Finally, run the following command:

```
> run_pyGenClean ^
      --conf conf.ini ^
      --bfile pyGenClean_test_data\1000G_EUR-MXL_Human610-Quad-v1_H
```

Results

Valuable information will be shown on the command prompt. Once the program has finished, the results are in the new directory `data_clean_up.date_time` where `date` is the current date, and `time` is the time when the program started.

Here are the new directory structure, with only the files you might be interested in:

- `data_clean_up.data_time\`
    - `pyGenClean.log`
    - `automatic_report.tex`
    - `excluded_markers.txt`
    - `excluded_samples.txt`
    - `1_check_ethnicity\`

    - ethnicity.before.png
    - ethnicity.outliers.png
    - ethnicity.outliers
    - ethnicity.population_file_outliers
  - 2_sex_check\
    - sexcheck.list_problem_sex

The file `pyGenClean.log` contain the information that was displayed in the console. The file `automatic_report.txt` contain the automatic report generated by *pyGenClean*. The files `excluded_markers.txt` and `excluded_samples.txt` contains the list of markers and samples, respectively, that were excluded from the dataset (with the reason).

1_check_ethnicity/

The first image in the first directory (*ethnicity.before.png*) shows the MDS values for each sample before outlier detection. The second image (*ethnicity.outliers.png*) shows the outliers that should be removed for further analysis. Finally, the file `ethnicity.outliers` include a list of samples that should be removed for further analysis. **The total number of outliers for this test should be exactly 63**. For more information about the results of this module, refer to Section *Ethnicity Module*.

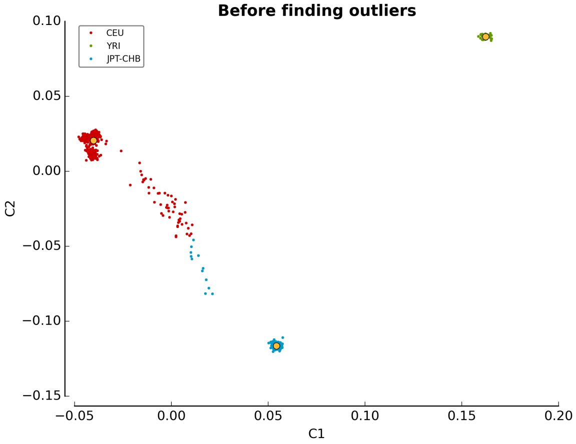

Fig. 2.3: ethnicity.before.png

2_sex_check/

In the second directory, there should be a file containing the list of samples with gender problem. **There should be exactly 4 samples with gender problem.** For more information about this module, refer to Section *Sex Check Module*.

If you want to compare your results with the expected ones, just download the files in the archive `pyGenClean_expected_results.tar.bz2`, available through http://statgen.org/downloads/pygenclean/. They were generated using Fedora 18 (64 bits) in about 20 minutes. You should at least compare the following files:

- `1_check_ethnicity`

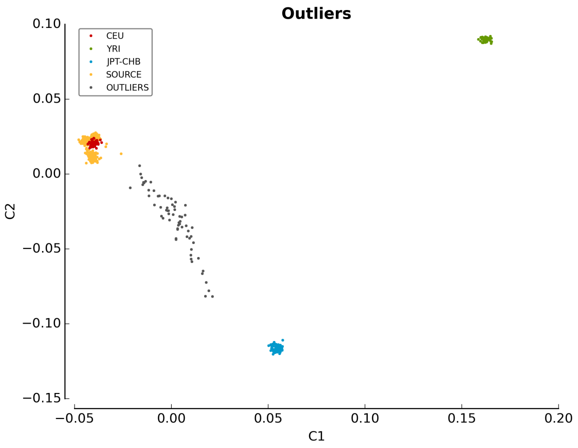

Fig. 2.4: ethnicity.outliers.png

- ethnicity.outliers
- ethnicity.population_file_outliers
- All the figures (they might be mirrored).

- 2_sex_check
  - sexcheck.list_problem_sex
  - sexcheck.list_problem_sex_ids

Automatic report

If LaTeX is installed, you can perform the following commands to compile the automatic report into a PDF file.

```
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
```

The following PDF report will be generated.

CHAPTER

THREE

INPUT FILES

To use `pyGenClean`, two sets of files are required: a set of genotype files and a configuration file (using the INI format).

3.1 Genotype files

The input files of the main program (`run_pyGenClean`) are either:

- PLINK's pedfile format (use `pyGenClean`'s `--file` option) consists of two files with the following extensions: `PED` and `MAP`.
- PLINK's transposed pedfile format (use `pyGenClean`'s `--tfile` option) consists of two files with the following extensions: `TPED` and `TFAM`.
- PLINK's binary pedfile format (use `pyGenClean`'s `--bfile` option) consists of three files with the following extensions: `BED`, `BIM` and `FAM`.

For more information about these file formats, have a look at PLINK's website, in the *Basic usage/data formats* section (http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml).

Warning: If the format used is the *transposed* one, the columns **must** be separated using **tabulations**, but alleles of each marker need to be separated by a single space.

To create this exact transposed pedfile format, you need to use the following PLINK's options:

- `--recode` to recode the file.
- `--transposed` to create an output file in the transposed pedfile format.
- `--tab` to use tabulations.

3.2 Configuration file

To customized `pyGenClean`, a basic configuration file is required. It tells which script to use in a specific order. It also sets the different options and input files, so that the analysis is easy to replicate or modify.

The configuration file uses the INI format. It consists of sections, led by a `[section]` header (contiguous integers which gives the order of the pipeline) and followed by customization of this particular part of the pipeline. Lines preceded by a # are comments and are not read by `pyGenClean`.

The following example first removes samples with a missing rate of 10% and more (section starting at line `1`), then removes markers with a missing rate of 2% and more (section starting at line `6`). Finally, it removes the samples with a missing rate of 2% and more (section starting at line `3`).

```
1  [1]
2  # Removes sample with a missing rate higher than 10%.
3  script = sample_missingness
4  mind = 0.1
5
6  [2]
7  # Removes markers with a missing rate higher than 2%.
8  script = snp_missingness
9  geno = 0.02
10
11 [3]
12 # Removes sample with a missing rate higher than 2%.
13 script = sample_missingness
14 mind = 0.02
```

For a more thorough example, complete configuration files are available for download at http://www.statgen.org and are explained in the *Configuration Files* section. For a list of available modules and standalone script, refer to the *List of Modules and their Options*.

3.2.1 List of Modules and their Options

The following sections show a list the available scripts that can be used in the configuration file, along with their options for customization.

Contamination

The name to use in the configuration file is `contamination` and the *List of options for the contamination script.* table shows its configuration.

Table 3.1: List of options for the **contamination** script.

Option	Type	Description
`--raw-dir`	`STRING`	Directory containing the raw data (one file per sample, where the name of the file (minus the extension) is the sample identification number.
`--colsample`	`STRING`	The sample column.
`--colmarker`	`STRING`	The marker column.
`--colbaf`	`STRING`	The B allele frequency column.
`--colab1`	`STRING`	The AB Allele 1 column.
`--colab2`	`STRING`	The AB Allele 2 column.
`--sge`		Use SGE for parallelization.
`--sge-walltime`	`STRING`	The walltime for the job to run on the cluster. Do not use if you are not required to specify a walltime for your jobs on your cluster (*e.g.* ‘`qsub -lwalltime=1:0:0`‘ on the cluster).
`--sge-nodes`	`INT`	The number of nodes and the number of processor per nodes to use (*e.g.* ‘`qsub -lnodes=X:ppn=Y`‘ on the cluster, where X is the number of nodes and Y is the number of processor to use. Do not use if you are not required to specify the number of nodes for your jobs on the cluster.
`--sample-per-run-for-sge`	`INT`	The number of sample to run for a single SGE job.

The name of the standalone script is `pyGenClean_check_contamination`.

Duplicated Samples

The name to use in the configuration file is `duplicated_samples` and the *List of options for the duplicated_samples script.* table shows its configuration.

Table 3.2: List of options for the **duplicated_samples** script.

Option	Type	Description
`--sample-completion-threshold`	`FLOAT`	The completion threshold to consider a replicate when choosing the best replicates and for creating the composite samples. [default: 0.9]
`--sample-concordance-threshold`	`FLOAT`	The concordance threshold to consider a replicate when choosing the best replicates and for creating the composite samples. [default: 0.97]

The name of the standalone script is `pyGenClean_duplicated_samples`.

Duplicated Markers

The name to use in the configuration file is `duplicated_snps` and the *List of options for the duplicated_snps script.* table shows its configuration.

Table 3.3: List of options for the **duplicated_snps** script.

Option	Type	Description
`--snp-completion-threshold`	`FLOAT`	The completion threshold to consider a replicate when choosing the best replicates and for composite creation. [default: 0.9]
`--snp-concordance-threshold`	`FLOAT`	The concordance threshold to consider a replicate when choosing the best replicates and for composite creation. [default: 0.98]
`--frequency_difference`	`FLOAT`	The maximum difference in frequency between duplicated markers [default: 0.05]

The name of the standalone script is `pyGenClean_duplicated_snps`.

Clean No Call and Only Heterozygous Markers

The name to use in the configuration file is `noCall_hetero_snps` and there are no customization possible.

The name of the standalone script is `pyGenClean_clean_noCall_hetero_snps`.

Sample Missingness

The name to use in the configuration file is `sample_missingness` and the *List of options for the sample_missingness script.* table shows its configuration.

Table 3.4: List of options for the **sample_missingness** script.

Option	Type	Description
`--mind`	`FLOAT`	The missingness threshold (remove samples with more than x percent missing genotypes). [Default: 0.100]

The name of the standalone script is `pyGenClean_sample_missingness`.

Marker Missingness

The name to use in the configuration file is `snp_missingness` and the *List of options for the snp_missingness script.* table shows its configuration.

Table 3.5: List of options for the **snp_missingness** script.

Option	Type	Description
`--geno`	`FLOAT`	The missingness threshold (remove SNPs with more than x percent missing genotypes). [Default: 0.020]

The name of the standalone script is `pyGenClean_snp_missingness`.

Sex Check

The name to use in the configuration file is `sex_check` and the *List of options for the sex_check script.* table shows its configuration.

Table 3.6: List of options for the **sex_check** script.

Option	Type	Description
`--femaleF`	`FLOAT`	The female F threshold. [default: < 0.300000]
`--maleF`	`FLOAT`	The male F threshold. [default: > 0.700000]
`--nbChr23`	`INT`	The minimum number of markers on chromosome 23 before computing Plink's sex check [default: 50]
`--gender-plot`		Create the gender plot (summarized chr Y intensities in function of summarized chr X intensities) for problematic samples. Not used by default.
`--sex-chr-intensities`	`FILE`	A file containing alleles intensities for each of the markers located on the X and Y chromosome for the gender plot.
`--gender-plot-format`	`STRING`	The output file format for the gender plot (png, ps, or pdf formats are available). [default: png]
`--lrr-baf`		Create the LRR and BAF plot for problematic samples. Not used by default.
`--lrr-baf-raw-dir`	`DIR`	Directory or list of directories containing information about every samples (BAF and LRR).
`--lrr-baf-format`	`STRING`	The output file format for the LRR and BAF plot (png, ps or pdf formats are available). [default: png]
`--lrr-baf-dpi`	`INT`	The pixel density of the figure(s) (DPI).

The name of the standalone script is `pyGenClean_sex_check`. If you want to redo the BAF and LRR plot or the gender plot, you can use the `pyGenClean_baf_lrr_plot` and `pyGenClean_gender_plot` scripts, respectively.

Plate Bias

The name to use in the configuration file is `plate_bias` and the *List of options for the plate_bias script.* table shows its configuration.

Table 3.7: List of options for the **plate_bias** script.

Option	Type	Description
`--loop-assoc`	`FILE`	The file containing the plate organization of each samples. Must contains three column (with no header): famID, indID and plateName.
`--pfilter`	`FLOAT`	The significance threshold used for the plate effect. [default: 1.0e-07]

The name of the standalone script is `pyGenClean_plate_bias`.

Heterozygous Haploid

The name to use in the configuration file is `remove_heterozygous_haploid` and there are no customization possible.

The name of the standalone script is `pyGenClean_remove_heterozygous_haploid`.

Related Samples

The name to use in the configuration file is `find_related_samples` and the *List of options for the find_related_samples script.* table shows its configuration.

Table 3.8: List of options for the **find_related_samples** script.

Option	Type	Description
`--genome-only`		Only create the genome file. Not selected by default.
`--min-nb-snp`	`INT`	The minimum number of markers needed to compute IBS values. [Default: 10000]
`--indep-pairwise`	`INT INT FLOAT`	Three numbers: window size, window shift and the r2 threshold. [default: ['50', '5', '0.1']]
`--maf`	`FLOAT`	Restrict to SNPs with MAF >= threshold. [default: 0.05]
`--ibs2-ratio`	`FLOAT`	The initial IBS2* ratio (the minimum value to show in the plot. [default: 0.8]
`--sge`		Use SGE for parallelization.
`--sge-walltime`	`STRING`	The time limit (for clusters). Do not use if you are not required to specify a walltime for your jobs on your cluster (e.g. `-lwalltime=1:0:0` on the cluster). Allow enough time for proper job completion.
`--sge-nodes`	`INT INT`	The number of nodes and the number of processor per nodes to use (e.g. `qsub -lnodes=X:ppn=Y` on the cluster, where X is the number of nodes and Y is the number of processor to use. Do not use if you are not required to specify the number of nodes for your jobs on the cluster. Allow enough ressources for proper job completion.
`--line-per-file-for-sge`	`INT`	The number of line per file for SGE task array. [default: 100]

The name of the standalone script is `pyGenClean_find_related_samples`. Even though randomly choosing a subset of related samples is done automatically, you can use the `pyGenClean_merge_related_samples` to perform it again.

Ethnicity

The name to use in the configuration file is `check_ethnicity` and the *List of options for the check_ethnicity script.* table shows its configuration.

Table 3.9: List of options for the **check_ethnicity** script.

Option	Type	Description
--skip-ref-pops		Perform the MDS computation, but skip the three reference panels.
--ceu-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the CEU population.
--yri-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the CEU population.
--jpt-chb-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the JPT-CHB population.
--min-nb-snp	FILE	The minimum number of markers needed to compute IBS values. [Default: 8000]
--indep-pairwise	INT INT FLOAT	Three numbers: window size, window shift and the r2 threshold. [default: ['50', '5', '0.1']]
--maf	INT	Restrict to SNPs with MAF >= threshold. [default: 0.05]
--sge		Use SGE for parallelization.
--sge-walltime	STRING	The time limit (for clusters). Do not use if you are not required to specify a walltime for your jobs on your cluster (e.g. `-lwalltime=1:0:0` on the cluster). Allow enough time for proper job completion.
--sge-nodes	INT INT	The number of nodes and the number of processor per nodes to use (e.g. `qsub -lnodes=X:ppn=Y` on the cluster, where X is the number of nodes and Y is the number of processor to use. Do not use if you are not required to specify the number of nodes for your jobs on the cluster. Allow enough ressources for proper job completion.
--ibs-sge-walltime	STRING	The time limit (for clusters) for the IBS jobs. Do not use if you are not required to specify a walltime for your jobs on your cluster (e.g. `-lwalltime=1:0:0` on the cluster). Allow enough time for proper job completion.
--ibs-sge-nodes	INT INT	The number of nodes and the number of processor per nodes to use for the IBS jobs (e.g. `qsub -lnodes=X:ppn=Y` on the cluster, where X is the number of nodes and Y is the number of processor to use. Do not use if you are not required to specify the number of nodes for your jobs on the cluster. Allow enough ressources for proper job completion.
--line-per-file-for-sge	INT	The number of line per file for SGE task array. [default: 100]
--nb-components	INT	The number of component to compute. [default: 10]
--outliers-of	STRING	Finds the outliers of this population. [default: CEU]
--multiplier	FLOAT	To find the outliers, we look for more than x times the cluster standard deviation. [default: 1.9]
--xaxis	STRING	The component to use for the X axis. [default: C1]
--yaxis	STRING	The component to use for the Y axis. [default: C2]
--format	STRING	The output file format (png, ps, pdf, or X11 formats are available). [default: png]
--title	STRING	The title of the MDS plot. [default: C2 in function of C1 - MDS]
--xlabel	STRING	The label of the X axis. [default: C1]
--ylabel	STRING	The label of the Y axis. [default: C2]
--create-scree-plot		Computes Eigenvalues and creates a scree plot.

The name of the standalone script is pyGenClean_check_ethnicity. If you want to redo the outlier detection using a different multiplier, have a look at the pyGenClean_find_outliers script. If you want to redo any MDS plot, have a look at the pyGenClean_plot_MDS script. If you want to compute the *Eigenvectors* using the smartpca tool, have a look at the pyGenClean_plot_eigenvalues script.

Minor Allele Frequency of Zero

The name to use in the configuration file is flag_maf_zero and there are no customization possible.

The name of the standalone script is pyGenClean_flag_maf_zero.

Hardy Weinberg Equilibrium

The name to use in the configuration file is flag_hw and the *List of options for the flag_hw script.* table shows its configuration.

Table 3.10: List of options for the **flag_hw** script.

Option	Type	Description
--hwe	FLOAT	The Hardy-Weinberg equilibrium threshold. [default: 1e-4]

The name of the standalone script is pyGenClean_flag_hw.

Subsetting the Data

The name to use in the configuration file is subset and the *List of options for the subset script.* table shows its configuration.

Table 3.11: List of options for the **subset** script.

Option	Type	Description
--exclude	FILE	A file containing SNPs to exclude from the data set.
--extract	FILE	A file containing SNPs to extract from the data set.
--remove	FILE	A file containing samples (FID and IID) to remove from the data set.
--keep	FILE	A file containing samples (FID and IID) to keep from the data set.

The name of the standalone script is pyGenClean_subset_data.

Comparison with a Gold Standard

The name to use in the configuration file is compare_gold_standard and the *List of options for the compare_gold_standard script.* table shows its configuration.

Table 3.12: List of options for the **compare_gold_standard** script.

Option	Type	Description
`--gold-bfile`	`FILE`	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the Gold Standard .
`--same-samples`	`FILE`	A file containing samples which are present in both the gold standard and the source panel. One line by identity and tab separated. For each row, first sample is Gold Standard, second is source panel.
`--source-manifest`	`FILE`	The illumina marker manifest.
`--source-alleles`	`FILE`	A file containing the source alleles (TOP). Two columns (separated by tabulation, one with the marker name, the other with the alleles (separated by space). No header.
`--sge`		Use SGE for parallelization.
`--do-not-flip`		Do not flip SNPs. WARNING: only use this option only if the Gold Standard was generated using the same chip (hence, flipping is unnecessary).
`--use-marker-names`		Use marker names instead of (chr, position). WARNING: only use this options only if the Gold Standard was generated using the same chip (hence, they have the same marker names).

The name of the standalone script is `pyGenClean_compare_gold_standard`.

CHAPTER

FOUR

INFORMATION ABOUT THE PROTOCOL

The following sections describe the proposed protocol and provides information about which file should be looked for quality control. Finally, configuration files (with all available parameters) are given.

4.1 Proposed Protocol

4.1.1 Contamination Module

Note: Input files:

- PLINK binary pedfiles (BED, BIM and FAM)

1. Examine the output file named `contamination.bafRegress`. It will contain the contamination estimates (along with confidence values). Usually, an estimate higher than 0.01 means possible contamination.
2. The automatic report will contain the list of samples with possible contamination (*i.e.* an estimate value higher than 0.01).

4.1.2 Preprocessing Steps

- Remove SNPs without chromosomal and physical position (chromosome and position of 0).
- Remove INDELs (markers with alleles `I` or `D`).
- Determine if there are duplicated samples. These samples must have exactly the same family (`FID`) and individual (`IID`) identification to be treated as duplicated samples by the *Duplicated Samples Module*. PLINK's option `--update-ids` could be used.
- If input is a transposed pedfile, be sure to use PLINK's option `--tab` to produce the appropriate file format.
- For the *Plate Bias Module*, a text file explaining the plate distribution of each sample must be provided using the option `--loop-assoc` in the configuration file. The following columns are required (in order, without a header):
  - the family identification;
  - the individual identification;
  - and the plate identification.
- Produce parameter files (see the *Configuration Files* for details about parameter file).
- To launch the analysis consult the section *How to Run the Pipeline*.

4.1.3 Duplicated Samples Module

Note: Input files:

- PLINK transposed pedfiles from the *Preprocessing Steps*.

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `dup_samples.diff` to evaluate if some samples have many discordant genotypes (this could indicate a possible samples mix up). To identify discordant samples, use the following command line:

   ```
   $ cut -f4 dup_samples.diff | sort -k1,1 | uniq -c
   $ cut -f5 dup_samples.diff | sort -k1,1 | uniq -c
   ```

   If samples are present more than 10,000 times (for 2.5E-6 SNPs) this could indicate a sample mix up.
3. Examine `dup_samples.not_good_enough` to determine if samples have a concordance rate below the threshold set by the user. These samples **are present** in the `dup_samples.final.tfam` if they are the chosen ones.
4. Examine `dup_samples.summary` to evaluate completion rate and concordance between the replicates of potentially problematic samples.
5. Examine `dup_samples.concordance` file for the problematic samples; this could help to determine which sample is the discordant replicate.
6. If a sample appears problematic rename it and keep it in the analysis to determine if it is a duplicate of another sample (mix up) with the related sample module.

If necessary, samples present in the `dup_samples.not_good_enough` file can be removed from the data set with the subset module (see the *First Subset Module (optional)*). If not, proceed to the *Duplicated Markers Module*).

4.1.4 First Subset Module (optional)

Note: Input files:

From the *Duplicated Samples Module*:

- `dup_samples.final.tfam`
- `dup_samples.final.tped`

1. Extract the family (`FID`) and individual (`IID`) identification from `dup_samples.not_good_enough` with the following command line:

   ```
   $ cut -f3,4 dup_samples.not_good_enough | sed "1d" | sort -k1,1 \
   >     | uniq > samples_to_remove
   ```

4.1.5 Duplicated Markers Module

Note: Input files:

From the *Duplicated Samples Module*:

- dup_samples.final.tfam
- dup_samples.final.tped

or from the *First Subset Module (optional)*:

- subset.bed
- subset.bim
- subset.fam

1. Examine the log to confirm options used and to detect any problems occurring while running the script
2. Examine dup_snps.duplicated_marker_names to detect SNPs with exactly the same name but mapping to different chromosomal location. (This file is not produce if no duplicated marker names are identified).
3. Determine the number of duplicated SNPs merged (same allele, same frequency, etc). SNPs merged were removed and are listed in the file dup_snps.removed_duplicates. Number of lines in this file corresponds to number of SNPs merged. SNPs not merged and reasons why (*e.g.* homo_hetero, diff_frequency, homo_flip, etc.) are present in file dup_snps.problems.
4. SNPs with concordance rate below the threshold are present in dup_snps.not_good_enough. To have the list of those SNPs:

```
$ grep -w concordance dup_snps.not_good_enough | cut -f1 \
>     > SNP_with_low_concordance_rate
```

If necessary, use the subset option in the configuration file to remove the low concordance rate SNPs (see the *Second Subset Module (optional)*).

4.1.6 Second Subset Module (optional)

Note: Input files:

From the *Duplicated Markers Module*:

- dup_snps.final.tfam
- dup_snps.final.tped

- Extract SNPs with concordance rate below the threshold set by the user with the command line

```
$ grep -w concordance dup_snps.not_good_enough | cut -f1 \
>     > SNP_with_low_concordance_rate
```

4.1.7 Clean No Call and Only Heterozygous Markers Module

Note: Input files:

From the *Duplicated Markers Module*:

- dup_snps.final.tfam
- dup_snps.final.tped

or from the *Second Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. SNPs removed because they are failed are listed in `clean_noCall_hetero.allFailed`.
3. SNPs removed because they are all heterozygous are listed in `clean_noCall_hetero.allHetero`.

4.1.8 Sample Missingness Module (mind 0.1)

Note: Input files:

From the *Clean No Call and Only Heterozygous Markers Module*:

- `clean_noCall_hetero.tfam`
- `clean_noCall_hetero.tped`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Individuals removed because they did not pass the completion rate threshold are listed in `clean_mind.irem`.

4.1.9 Marker Missingness Module

Note: Input files:

From the *Sample Missingness Module (mind 0.1)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. SNPs removed because they did not pass the completion rate threshold are listed in `clean_geno.removed_snps`.

4.1.10 Sample Missingness Module (mind 0.02)

Note: Input files:

From the *Marker Missingness Module*:

- `clean_geno.bed`
- `clean_geno.bim`

- clean_geno.fam

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Individuals removed because they did not pass the completion rate threshold are listed in clean_mind.irem.

4.1.11 Sex Check Module

Note: Input files:

From *Sample Missingness Module (mind 0.02)*:

- clean_mind.bed
- clean_mind.bim
- clean_mind.fam

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Examine sexcheck.list_problem_sex, it contains all individuals identified by PLINK as having gender problem.
4. Examine sexcheck.chr23_recodeA.raw.hetero to determine heterozygosity on the X chromosome of problematic samples. Consanguineous females may have low heterozygosity on the X chromosome. If many genotyped SNPs are rare, heterozygosity may also be low.
5. Examine sexcheck.chr24_recodeA.raw.noCall to determine the number of Y markers with missing calls. Females have low number of genotypes for Y chromosome markers (high values of missing calls), but is often not equal to 0 probably because some Y markers come from pseudo autosomal regions. Column nbGeno is the total number of genotypes check and nbNoCall is the number of genotypes with missing calls on chromosome Y. Males should have low values in this column while females have higher number of missing calls but not equal to the total number of genotypes tested.

If probe intensities from X and Y chromosomes are available and the gender plot has been created:

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine sexcheck.png to detect any individuals in the XXY or X0 regions, females in the male cluster and males in the female cluster (see the *Gender plot* figure). Confirm if possible the gender problems identified with the previous sex check problem step.

If intensities file for each sample are available and the BAF and LRR plot has been created:

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine sexcheck_sample-id_lrr_baf.png for each sample. Usually, females have LRR values around 0 (between -0.5 and 0.5) while males have LRR values between -0.5 and -1. Females have three lines on BAF graphics: one at 1 (homozygous for the B allele), one at 0.5 (heterozygous AB) and one at 0 (homozygous for the A allele). Males have two lines: one at 1 (homozygous for the B allele) and one a at 0 (homozygous for the A allele). For more details, see the *The Plots* section of the *Sex Check Module*.

Keep individuals identified with gender problem until the *Related Samples Module* (mix up of samples could be resolved at this step).

4.1.12 Plate Bias Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

or if subset option is used to remove SNPs from `nof` file (see below):

- `subset.bed`
- `subset.bim`
- `subset.fam`

1. Verify if there is a `nof` file produce by PLINK when the input files for this step were produced (from the *Sample Missingness Module (mind 0.02)*). The `nof` contains SNPs with no founder genotypes observed. If so, remove the SNPs present in the `nof` file using the subset tool before launching the plate bias analysis. Those SNPs, if they are not removed will produced an error message when PLINK performs the `loop-assoc` analysis and the following message will be present in PLINK's log file `plate_bias.log`: "`ERROR: FEXACT error 3`". SNPs on chromosome 24 could also produce this error.
2. Examine the log to confirm options used and to detect any problems occurring while running the script.
3. Examine `plate_bias.log` to detect any problem at this step.
4. The `plate_bias.significant_SNPs.txt` file contains a list of SNPs with P value below the threshold. Care should be taken with those SNPs if significant results are obtained in association tests. These SNPs are NOT removed from the data set, only flagged.
5. Low MAF can explain part of plate bias. Examine the output file `plate_bias.significant_SNPs.frq` to determine if SNPs have low MAF. Other reasons explaining plate bias are relatedness or ethnicity of individuals assign to the same plates and none of them on other plates.

4.1.13 Related Samples Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. File `ibs.pruning_0.1.prune.in` contains the list of uncorrelated SNPs used for the IBS analysis
3. Examine `ibs.related_individuals_z1.png` and `ibs.related_individuals_z2.png` to detect if there are samples in the parent-child, duplicated samples, first degree relative and second degree relative areas. (see *Z1 in function of IBS2 ratio* and *Z2 in function of IBS2 ratio* plots).

4. File `ibs.related_individuals` lists pairs of related individuals. Index column indicates group of related samples. Status column indicated the probable link between pair of individuals based on Z_0, Z_1 and Z_2 values (see the *IBD allele sharing values* table [for which Z values are approximation] or `RelatedSamples.find_related_samples.extractRelatedIndividuals()` function for thresholds).
5. If there are known duplicated samples, examine `ibs.related_individuals` to determine if they were identified correctly, if not this could indicate a possible samples mix up.
6. File `ibs.choosen_related_individuals` contains a list of related samples to keep. One related sample from the pair is randomly selected. If there are a group of related individuals, one sample in randomly selected from the group. All non selected samples are listed in `ibs.discarded_related_individuals` and should be removed from the analysis at a later stage.

Table 4.1: IBD allele sharing values

Relationship	k_0	k_1	k_2	Coancestry $\theta = 1/2k_2 + 1/4k_1$
Unrelated	1	0	0	0
Identical twins	0	0	1	1/2
Parent-child	0	1	0	1/4
Full siblings	1/4	1/2	1/4	1/4
Half siblings	1/2	1/2	0	1/8
Uncle nephew	1/2	1/2	0	1/8
Grandparent grandchild	1/2	1/2	0	1/8
Double first cousins	9/16	3/8	1/16	1/8
First cousins	3/4	1/4	0	1/16

4.1.14 Ethnicity Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. File `ethnic.ibs.pruning_0.1.prune.in` contains the list of uncorrelated SNPs used for the MDS analysis.
3. File `ethnic.mds.mds` contains the list of principale components as calculated by PLINK.
4. Examine `ethnicity.mds.png`, `ethnicity.before.png`, `ethnicity.after.png` and `ethnicity.outliers.png` to detect samples outside the selected cluster (see *The Plots* generated from the *Ethnicity Module* for more information).

   If there are too many outliers still present in the data set (*i.e.* radius is too large), analysis can be redone using the `pyGenClean_find_outliers` standalone script, using a different value for `--multiplier`. For more information, refer to the *Finding Outliers* section of the *Ethnicity Module*.
5. Samples outside the selected cluster are listed in `ethnicity.outliers`. If necessary those samples could be removed at a later stage with the subset option.

4.1.15 Third Subset Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- clean_mind.bed
- clean_mind.bim
- clean_mind.fam

Use the subset module to remove samples with gender problems (the *Sex Check Module*), outliers from the ethnicity cluster (the *Ethnicity Module*), related samples (the *Related Samples Module*) and any other samples that need to be removed from the data set.

- To produces a file containing all the samples to remove from the dataset:

```
$ cat sexcheck.list_problem_sex_ids ibs.discarded_related_individuals \
>     ethnicity.outliers > samples_to_remove.txt
```

  One sample may be removed for more than one reason, hence be present more than one time in the final samples_to_remove.txt file. This is not an issue for this step.

4.1.16 Heterozygote Haploid Module

Note: Input files:

From the *Third Subset Module*:

- subset.bed
- subset.bim
- subset.fam

Samples with gender problems **must have been removed before** performing this module.

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine without_hh_genotypes.log to detect any problem at this step.

Number of heterozygous haploid genotypes set to missing are indicated in without_hh_genotypes.log file.

4.1.17 Minor Allele Frequency of Zero Module

Note: Input files:

From the *Heterozygote Haploid Module*:

- without_hh_genotypes.bed
- without_hh_genotypes.bim
- without_hh_genotypes.fam

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `flag_maf_0.log` to detect any problem at this step.
3. The file `flag_maf_0.na_list` contains a list of SNPs with minor allele frequency of 0.

If necessary, use subset module to remove SNPs with minor allele frequency of 0, since they were only flagged using the *Fourth Subset Module (optional)*.

4.1.18 Fourth Subset Module (optional)

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `subset.log` to detect any problem at this step.

4.1.19 Hardy Weinberg Equilibrium Module

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

or from the *Fourth Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `flag_hw.threshold_1e-4.log` and `flag_hw.threshold_Bonferroni.log` to detect any problem at this step.
3. The files `flag_hw.snp_flag_threshold_Bonferroni` and `flag_hw.snp_flag_threshold_1e-4` contain lists of SNPs with P value below Bonferroni and below 1×10^{-4} threshold, respectively.

The markers are only flagged using this module. If you want to remove those markers, have a look at the *Fifth Subset Module (optional)*.

4.1.20 Fifth Subset Module (optional)

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

or from the *Fourth Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subsert.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `subset.log` to detect any problem at this step.

4.2 Configuration Files

Two default configuration files are available to run the proposed protocol. Before using them, be sure to follow the *Preprocessing Steps* described in the *Proposed Protocol* section.

Note that lines starting with a `#` are comments, and are not used by the pipeline. The default parameters were commented out, and could be uncommented to change their values.

4.2.1 First Configuration File

This file should be use with the original dataset as input. Only change the `loop-assoc` file name in the `plate_bias` section (`[8]`) and the reference population files (`ceu-bfile`, `yri-bfile` and `jpt-chb-bfile` in the `check_ethnicity` section (`[10]`). Those last three datasets are provided and can be downloaded at http://www.statgen.org.

If you want to generate the gender and BAF and LRR plots, you will require to provide the intensities (`sex-chr-intensities` and `lrr-baf-raw-dir` in the `sex_check` section (`[7]`) after uncommenting the required options).

```
# This is the first part of example configuration files for performing efficient
# data clean up. All commented out parameters are those that are used by
# default.


[1]
# ###############################################################################
# Checks sample contamination using the bafRegress tool
# (http://genome.sph.umich.edu/wiki/BAFRegress). Field name can be modify using
# options (as describe below).
# ###############################################################################

script = contamination
```

```
raw-dir = /PATH/TO/DIRECTORY/CONTAINING/INTENSITIES.txt
# colsample = Sample Name
# colmarker = SNP Name
# colbaf = B Allele Freq
# colab1 = Allele1 - AB
# colab2 = Allele2 - AB
# sge
# sge-walltime = WRITE WALLTIME ONLY IF REQUIRED
# sge-nodes = WRITE NB NODES AND NB PROCESSOR PER NODE ONLY IF REQUIRED
# sample-per-run-for-sge = 30



[2]
# ##############################################################################
# Checks missing rate and pairwise concordance of duplicated samples. Duplicated
# samples should have same family and individual identification numbers. The
# names can be modified directly in the transposed pedfile.
# ##############################################################################

script = duplicated_samples
# sample-completion-threshold = 0.9
# sample-concordance-threshold = 0.97



[3]
# ##############################################################################
# Checks missing rate and pairwise concordance of duplicated markers. Duplicated
# markers are found by looking at their chromosomal position. No modification of
# the transposed bedfile is required.
# ##############################################################################

script = duplicated_snps
# snp-completion-threshold = 0.9
# snp-concordance-threshold = 0.98
# frequency_difference = 0.05



[4]
# ##############################################################################
# Finds and removes markers which have a missing rate of 100% or markers (not
# located on mitochondrial chromosome) that have a heterozygosity rate of 0%.
# ##############################################################################

script = noCall_hetero_snps



[5]
# ##############################################################################
# Removes sample with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 10% missing rate as samples with
# a missing rate of 2% will be later removed.
# ##############################################################################

script = sample_missingness
```

```
# mind = 0.1



[6]
# ##############################################################################
# Removes markers with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 2% missing rate.
# ##############################################################################

script = snp_missingness
# geno = 0.02



[7]
# ##############################################################################
# Removes sample with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 2% missing rate.
# ##############################################################################

script = sample_missingness
mind = 0.02



[8]
# ##############################################################################
# Using PLINK, finds samples with gender issues, according to heterozygosity
# rate on the X chromosome. If you want to produce a gender plot, you need to
# uncomment the "gender-plot" option and provide a file containing marker
# intensities on the X and Y chromosomes. If you want to produce a BAF and LRR
# plot, you need to uncomment the "lrr-baf" option and provide a directory
# containing the BAF and LRR values of each marker on the X and Y chromosomes
# (one file per sample).
# ##############################################################################

script = sex_check
# femaleF = 0.3
# maleF = 0.7
# nbChr23 = 50
# gender-plot
# sex-chr-intensities = /PATH/TO/FILE/CONTAINING/INTENSITIES_FILE.txt
# gender-plot-format = png
# lrr-baf
# lrr-baf-raw-dir = /PATH/TO/DIRECTORY/CONTAINING/BAF_LRR_FILES.txt
# lrr-baf-format = png
# lrr-baf-dpi = 300



[9]
# ##############################################################################
# Using PLINK, performs a plate bias analysis, using a p value threshold of
# 1.0e-7.
# ##############################################################################

script = plate_bias
```

```
loop-assoc = /PATH/TO/FILE/CONTAINING/PLATE_INFORMATION.txt
# pfilter = 1.0e-07



[10]
# ##############################################################################
# Checks for related individual and randomly keeps one of each related group. If
# you have a server with a DRMAA-compliant distributed resource management
# system, you can uncomment the "sge" and the "line-per-file-for-sge" options,
# to run this step in parallel.
# ##############################################################################

script = find_related_samples
# min-nb-snp = 10000
# indep-pairwise = 50 5 0.1
# maf = 0.05
# ibs2-ratio = 0.8
# sge
# line-per-file-for-sge = 100
# sge-walltime = WRITE WALLTIME ONLY IF REQUIRED
# sge-nodes = WRITE NB NODES AND NB PROCESSOR PER NODE ONLY IF REQUIRED



[11]
# ##############################################################################
# Using PLINK, computes the MDS value of each sample, and using three reference
# populations (CEU, YRI and JPT-CHB), finds outliers of one of those three
# reference population. You might want to skip the reference population using
# the "skip-ref-pops" option. You might need to change the "multiplier" option
# to be more or less stringent, according to you dataset. If you have a server
# with a DRMAA-compliant distributed resource management system, you can
# uncomment the "sge" and the "line-per-file-for-sge" options, to run this step
# in parallel.
# ##############################################################################

script = check_ethnicity
ceu-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/CEU_population
yri-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/YRI_population
jpt-chb-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/JPT-CHB_population
# skip-ref-pops
# min-nb-snp = 8000
# indep-pairwise = 50 5 0.1
# maf = 0.05
# sge
# line-per-file-for-sge = 100
# nb-components = 10
# outliers-of = CEU
# multiplier = 1.9
# xaxis = C1
# yaxis = C2
# format = png
# title = "C2 in function of C1 - MDS"
# xlabel = C1
# ylabel = C2
# create-scree-plot
# scree-plot-title "TITLE OF THE PLOT"
# sge-walltime = WRITE WALLTIME ONLY IF REQUIRED
# sge-nodes = WRITE NB NODES AND NB PROCESSOR PER NODE ONLY IF REQUIRED
# ibs-sge-walltime = WRITE WALLTIME ONLY IF REQUIRED
# ibs-sge-nodes = WRITE NB NODES AND NB PROCESSOR PER NODE ONLY IF REQUIRED
```

4.2.2 Second Configuration File

This configuration file should be run after the *First Configuration File* and with the output of the second sample missingness section (`[6]` in the *First Configuration File*).

A file containing the samples and markers to be removed should be created using the output of the `sex_check`, `find_related_samples`, `check_ethnicity` and `plate_bias` sections of the *First Configuration File*.

```
# This is the second part of example configuration files for performing
# efficient data clean up. All commented out parameters are those that are used
# by default.

# The input file should be the output file of the second sample missigness step
# (which is the one that has been used by any of these scripts):
#     - sex_check
#     - find_related_samples
#     - check_ethnicity
#     - plate_bias

# Note that the final usable dataset is the one located in the directory where
# "remove_heterozygous_haploid" was run (which is the one that has been used by
# any of these scripts):
#     - flag_maf_zero
#     - flag_hw
# Hence, if you want to remove the flagged markers, you should use
# pyGenClean_subset_data on markers in the "flag_maf_zero" and "flag_hw"
# directories using the PLINK's binary file located in
# "remove_heterozygous_haploid".

[12]
# ##########################################################################
# After manually checking that everything went fine in the previous steps, you
# need to create a list of samples to remove from steps [7] to [10] and a list
# of markers to exclude from steps [6]. Just create a file containing family and
# individual identification numbers for all those samples to remove. Note that
# the two options 'reason-marker' and 'reason-sample' are for the automatic
# report generated after the analysis.
# ##########################################################################

script = subset
reason-marker = reason for marker exclusion
reason-sample = reason for sample exclusion
remove = /PATH/TO/FILE/CONTAINING/ALL_SAMPLES_FROM_PREVIOUS_STEPS_TO_REMOVE.txt
exclude = /PATH/TO/FILE/CONTAINING/ALL_MARKERS_FROM_PREVIOUS_STEPS_TO_EXCLUDE.txt



[13]
# ##########################################################################
# Removes heterozygous haploid genotypes from the dataset.
# ##########################################################################
```

```

script = remove_heterozygous_haploid



[14]
# ##########################################################################
# Flags uninformative markers (with a MAF of 0). This step only flag markers.
# You might want to exclude them later on.
# ##########################################################################

script = flag_maf_zero



[15]
# ##########################################################################
# Flags markers that fail HWE test for a p value of 1e-4 and after Bonferroni
# correction. This step only flag markers. You might want to exclude them later
# on.
# ##########################################################################

script = flag_hw
# hwe = 1e-4
```

4.3 How to Run the Pipeline

Warning: Before using *pyGenClean*, be sure to activate the appropriate Python virtual environment (refer to the Linux *virtualenv* or *conda* installation section, or the *Windows* installation section for more information.

Modify the *First Configuration File* so that it suits your needs. After following the *Preprocessing Steps* described in the *Proposed Protocol* section, run the following command:

```
$ run_pyGenClean \
>     --conf configuration_example_1_of_2.ini \
>     --tfile /PATH/TO/ORIGINAL/DATASET_PREFIX
```

While the protocol is running, check the outputs according to the *Proposed Protocol*. If there are any problems, interrupt the analysis and make the required modifications. The completed steps can be skipped by commenting them out, while using the last output dataset as the input one for the steps that need to be done again.

Once everything was checked, locate the samples and the markers that need to be removed. For example, if the output directory from the first dataset is `data_clean_up.YYYY-MM-DD_HH.MM.SS`, the following command will help you:

```
$ output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
$ cat $output_dir/7_sex_check/sexcheck.list_problem_sex_ids \
>     $output_dir/9_find_related_samples/ibs.discarded_related_individuals \
>     $output_dir/10_check_ethnicity/ethnicity.outliers \
>     > samples_to_remove.txt
```

Then, modify the first `subset` section in the *Second Configuration File* so that it reads:

```
[11]
script = subset
remove = samples_to_remove.txt
exclude = data_clean_up.YYYY-MM-DD_HH.MM.SS/8_plate_bias/plate_bias.significant_SNPs.
↪txt
```

Once everything was checked, run the following command to finish the data clean up pipeline:

```
output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
run_pyGenClean \
    --conf configuration_example_2_of_2.ini \
    --bfile $output_dir/6_sample_missingness/clean_mind
```

If you want to removed the markers that were flagged in the `flag_maf_zero` and `flag_hw` section, performed the following commands (using the newly created output directory `data_clean_up.YYYY-MM-DD_HH.MM.SS`):

```
$ output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
$ cat $output_dir/13_flag_maf_zero/flag_maf_0.list \
>     $output_dir/14_flag_hw/flag_hw.snp_flag_threshold_1e-4 \
>     > markers_to_exclude.txt
$ pyGenClean_subset_data \
>     --ifile $output_dir/14_remove_heterozygous_haploid/without_hh_genotypes \
>     --is-bfile \
>     --exclude markers_to_exclude.txt \
>     --out final_dataset
```

4.4 Automatic Report

The `pyGenClean` module creates an automatic report using *LaTeX*. Each of the output directory will contain a final report (named `automatic_report.tex`). The report, once compiled into the PDF format, includes various quality metrics. To compile the report, perform the following commands:

```
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
$ pdflatex automatic_report.tex
```

Note: Executing the command three times insure that the links in the automatic report are properly formed.

4.4.1 Report merging

On a typical data clean up pipeline, multiple directories will be created (one for each of the parts of the pipeline). A script is provided to merge all those reports (one per `data_clean_up.YYYY-MM-DD_HH.MM.SS` directory) into a single report. Here is the usage of this script:

```
$ pyGenClean_merge_reports --help
usage: pyGenClean_merge_reports [-h] --qc-dir DIR [DIR ...]
                                [--report-title TITLE]
                                [--report-author AUTHOR]
                                [--report-number NUMBER]
                                [--report-background BACKGROUND]
                                [--out-dir FILE]
```

```
Merges automatic reports from other pyGenClean runs.

optional arguments:
  -h, --help            show this help message and exit

Input:
  --qc-dir DIR [DIR ...]
                        A list of directory containing pyGenClean runs.

Report Options:
  --report-title TITLE  The report title. [default: Genetic Data Clean Up]
  --report-author AUTHOR
                        The current project number. [default: pyGenClean]
  --report-number NUMBER
                        The current project author. [default: Simple Project]
  --report-background BACKGROUND
                        Text of file containing the background section of the
                        report.

Output Directory:
  --out-dir FILE        The name of the directory that will contain the final
                        report. [default: pyGenClean_report]
```

To execute the report merging procedure, perform the following command:

```
$ pyGenClean_merge_reports --qc-dir data_clean_up.*
```

Note: It is possible to change the title, the author, the number or the background of the automatic report by using the `--report-title`, `--report-author`, `--report-number` or `--report-background` options, respectively.

Once again, to compile the final report, perform the following command:

```
$ pdflatex pyGenClean_report.tex
$ pdflatex pyGenClean_report.tex
$ pdflatex pyGenClean_report.tex
```

CHAPTER FIVE

THE ALGORITHM

All the functions used for this project are shown and explained in the following section:

5.1 Main pyGenClean pipeline

Here is the usage of the main pipeline.

```
$ run_pyGenClean --help
usage: run_pyGenClean [-h] [-v] [--bfile FILE] [--tfile FILE] [--file FILE]
                      [--report-title TITLE] [--report-author AUTHOR]
                      [--report-number NUMBER]
                      [--report-background BACKGROUND] --conf FILE

Runs the data clean up (pyGenClean version 1.8.3).

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively).
  --tfile FILE          The input file prefix (will find the plink transposed
                        files by appending the prefix to the .tped and .tfam
                        files, respectively).
  --file FILE           The input file prefix (will find the plink files by
                        appending the prefix to the .ped and .fam files).

Report Options:
  --report-title TITLE  The report title. [default: Genetic Data Clean Up]
  --report-author AUTHOR
                        The current project number. [default: pyGenClean]
  --report-number NUMBER
                        The current project author. [default: Simple Project]
  --report-background BACKGROUND
                        Text of file containing the background section of the
                        report.

Configuration File:
  --conf FILE           The parameter file for the data clean up.
```

5.1.1 Algorithm

For more information about the algorithms, have a look below.

pyGenClean package

Subpackages

pyGenClean.Contamination package

For more information about how to use this module, refer to the *Contamination Module*.

Module contents

Submodules

pyGenClean.Contamination.contamination module

exception `pyGenClean.Contamination.contamination.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (`str`) – the message to print to the user before exiting.

`pyGenClean.Contamination.contamination.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.Contamination.contamination.`**`check_sample_files`**(*fam_filename*, *raw_dirname*)

Checks the raw sample files.

Parameters

- **`fam_filename`** (`str`) – the name of the FAM file.
- **`raw_dirname`** (`str`) – the name of the directory containing the raw file.

Returns the set of all the sample files that are compatible with the FAM file.

Return type set

`pyGenClean.Contamination.contamination.`**`create_extraction_file`**(*bim_filename*, *out_prefix*)

Creates an extraction file (keeping only markers on autosomes).

Parameters

- **`bim_filename`** (`str`) – the name of the BIM file.
- **`out_prefix`** (`str`) – the prefix for the output file.

`pyGenClean.Contamination.contamination.`**`main`**`(`*argString=None*`)`

The main function of the module.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1. Prints the options.
2. Compute frequency using Plink.
3. Runs bafRegress.

`pyGenClean.Contamination.contamination.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively).
`--raw-dir`	string	Directory containing the raw data (one file per sample, where the name of the file (minus the extension) is the sample identification number.
`--colsample`	string	The sample column.
`--colmarker`	string	The marker column.
`--colbaf`	string	The B allele frequency column.
`--colab1`	string	The AB Allele 1 column.
`--colab2`	string	The AB Allele 2 column.
`--out`	string	The prefix of the output files.
`--sge`	bool	Use SGE for parallelization.
`--sge-walltime`	string	The walltime for the job to run on the cluster. Do not use if you are not required to specify a walltime for your jobs on your cluster (e.g. ‘qsub -lwalltime=1:0:0’ on the cluster).
`--sge-nodes`	int	The number of nodes and the number of processor per nodes to use (e.g. ‘qsub -lnodes=X:ppn=Y’ on the cluster, where X is the number of nodes and Y is the number of processor to use. Do not use if you are not required to specify the number of nodes for your jobs on the cluster.
`--sample-per-run-for-sge`	int	The number of sample to run for a single SGE job.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.Contamination.contamination.`**`run_bafRegress`**`(`*filenames*, *out_prefix*, *extract_filename*, *freq_filename*, *options*`)`

Runs the bafRegress function.

Parameters

- **`filenames`** (*`set`*) – the set of all sample files.
- **`out_prefix`** (*`str`*) – the output prefix.
- **`extract_filename`** (*`str`*) – the name of the markers to extract.
- **`freq_filename`** (*`str`*) – the name of the file containing the frequency.
- **`options`** (*`argparse.Namespace`*) – the other options.

`pyGenClean.Contamination.contamination.`**`run_bafRegress_sge`**`(`*filenames*, *out_prefix*, *extract_filename*, *freq_filename*, *options*`)`

Runs the bafRegress function using SGE.

Parameters

- **`filenames`** (*`set`*) – the set of all sample files.
- **`out_prefix`** (*`str`*) – the output prefix.
- **`extract_filename`** (*`str`*) – the name of the markers to extract.
- **`freq_filename`** (*`str`*) – the name of the file containing the frequency.
- **`options`** (*`argparse.Namespace`*) – the other options.

`pyGenClean.Contamination.contamination.`**`run_plink`**`(`*in_prefix*, *out_prefix*, *extract_filename*`)`

Runs Plink with the geno option.

Parameters

- **`in_prefix`** (*`str`*) – the input prefix.
- **`out_prefix`** (*`str`*) – the output prefix.
- **`extract_filename`** – the name of the file containing markers to extract.
- **`extract_filename`** – str

`pyGenClean.Contamination.contamination.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.DupSNPs package

For more information about how to use this module, refer to the *Duplicated Markers Module*.

Module contents

Submodules

pyGenClean.DupSNPs.duplicated_snps module

exception `pyGenClean.DupSNPs.duplicated_snps.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.DupSNPs.duplicated_snps.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

pyGenClean.DupSNPs.duplicated_snps.**chooseBestSnps**(*tped*, *snps*, *trueCompletion*, *trueConcordance*, *prefix*)

Choose the best duplicates according to the completion and concordance.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **snps** (*dict*) – the position of the duplicated markers in the `tped`.
- **trueCompletion** (*numpy.array*) – the completion of each markers.
- **trueConcordance** (*dict*) – the pairwise concordance of each markers.
- **prefix** (*str*) – the prefix of the output files.

Returns a tuple containing the chosen indexes (`dict`) as the first element, the completion (`numpy.array`) as the second element, and the concordance (`dict`) as last element.

It creates two output files: `prefix.chosen_snps.info` and `prefix.not_chosen_snps.info`. The first one contains the markers that were chosen for completion, and the second one, the markers that weren't.

It starts by computing the completion of each markers (dividing the number of calls divided by the total number of genotypes). Then, for each of the duplicated markers, we choose the best one according to completion and concordance (see explanation in `DupSamples.duplicated_samples.chooseBestDuplicates()` for more details).

pyGenClean.DupSNPs.duplicated_snps.**computeFrequency**(*prefix*, *outPrefix*)

Computes the frequency of the SNPs using Plink.

Parameters

- **prefix** (*str*) – the prefix of the input files.
- **outPrefix** (*str*) – the prefix of the output files.

Returns a `dict` containing the frequency of each marker.

Start by computing the frequency of all markers using Plink. Then, it reads the output file, and saves the frequency and allele information.

pyGenClean.DupSNPs.duplicated_snps.**computeStatistics**(*tped*, *tfam*, *snps*)

Computes the completion and concordance of each SNPs.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped`.
- **tfam** (*list*) – a representation of the `tfam`
- **snps** (*dict*) – the position of the duplicated markers in the `tped`.

Returns a tuple containing the completion of duplicated markers (`numpy.array`) as first element, and the concordance (`dict`) of duplicated markers, as last element.

A marker's completion is compute using this formula (where G_i is the set of genotypes for the marker i):

$$Completion_i = \frac{||g \in G_i \textrm{ where } g \neq 0||}{||G_i||}$$

The pairwise concordance between duplicated markers is compute as follow (where G_i and G_j are the sets of genotypes for markers i and j, respectively):

$$Concordance_{i,j} = \frac{||g \in G_i \cup G_j \textrm{ where } g_i = g_j \neq 0||}{||g \in G_i \cup G_j \textrm{ where } g \neq 0||}$$

Hence, we only computes the numerators and denominators of the completion and concordance, for future reference.

Note: When the genotypes are not comparable, the function tries to flip one of the genotype to see if it becomes comparable.

`pyGenClean.DupSNPs.duplicated_snps.`**`createAndCleanTPED`**(*tped*, *tfam*, *snps*, *prefix*, *chosenSNPs*, *completion*, *concordance*, *snpsToComplete*, *tfamFileName*, *completionT*, *concordanceT*)

Complete a TPED for duplicated SNPs.

Parameters

- **`tped`** (*`numpy.array`*) – a representation of the `tped` of duplicated markers.
- **`tfam`** (*`list`*) – a representation of the `tfam`.
- **`snps`** (*`dict`*) – the position of duplicated markers in the `tped`.
- **`prefix`** (*`str`*) – the prefix of the output files.
- **`chosenSNPs`** (*`dict`*) – the markers that were chosen for completion (including problems).
- **`completion`** (*`numpy.array`*) – the completion of each of the duplicated markers.
- **`concordance`** (*`dict`*) – the pairwise concordance of the duplicated markers.
- **`snpsToComplete`** (*`set`*) – the markers that will be completed (excluding problems).
- **`tfamFileName`** (*`str`*) – the name of the original `tfam` file.
- **`completionT`** (*`float`*) – the completion threshold.
- **`concordanceT`** (*`float`*) – the concordance threshold.

Returns a tuple containing the new `tped` after completion (`numpy.array` as the first element, and the index of the markers that will need to be rid of (`set`) as the last element.

It creates three different files:

- **`prefix.zeroed_out`: contains information about markers and samples** where the genotyped was zeroed out.
- **`prefix.not_good_enough`: contains information about markers that were** not good enough to help in completing the chosen markers (because of concordance or completion).
- **`prefix.removed_duplicates`: the list of markers that where used for** completing the chosen one, hence they will be removed from the final data set.

Cycling through every genotypes of every samples of every duplicated markers, checks if the genotypes are all the same. If the chosen one was not called, but the other ones were, then we complete the chosen one with the genotypes for the others (assuming that they are all the same). If there is a difference between the genotypes, it is zeroed out for the chosen marker.

`pyGenClean.DupSNPs.duplicated_snps.`**`createFinalTPEDandTFAM`**(*tped*, *toReadPrefix*, *prefix*, *snpToRemove*)

Creates the final TPED and TFAM.

Parameters

- **`tped`** (*`numpy.array`*) – a representation of the `tped` of duplicated markers.

- **toReadPrefix** (*str*) – the prefix of the unique files.
- **prefix** (*str*) – the prefix of the output files.
- **snpToRemove** (*set*) – the markers to remove.

Starts by copying the unique markers' `tfam` file to `prefix.final.tfam`. Then, it copies the unique markers' `tped` file, in which the chosen markers will be appended.

The final data set will include the unique markers, the chosen markers which were completed, and the problematic duplicated markers (for further analysis). The markers that were used to complete the chosen ones are not present in the final data set.

`pyGenClean.DupSNPs.duplicated_snps.`**findUniques**(*mapF*)

Finds the unique markers in a MAP.

Parameters **mapF** (*list*) – representation of a `map` file.

Returns a `dict` containing unique markers (according to their genomic localisation).

`pyGenClean.DupSNPs.duplicated_snps.`**flipGenotype**(*genotype*)

Flips a genotype.

Parameters **genotype** (*set*) – the genotype to flip.

Returns the new flipped genotype (as a `set`)

```
>>> flipGenotype({"A", "T"})
set(['A', 'T'])
>>> flipGenotype({"C", "T"})
set(['A', 'G'])
>>> flipGenotype({"T", "G"})
set(['A', 'C'])
>>> flipGenotype({"0", "0"})
Traceback (most recent call last):
    ...
ProgramError: 0: unkown allele
>>> flipGenotype({"A", "N"})
Traceback (most recent call last):
    ...
ProgramError: N: unkown allele
```

`pyGenClean.DupSNPs.duplicated_snps.`**getIndexOfHeteroMen**(*genotypes, menIndex*)

Get the indexes of heterozygous men.

Parameters

- **genotypes** (*numpy.array*) – the genotypes of everybody.
- **menIndex** (*numpy.array*) – the indexes of the men (for the genotypes).

Returns a `numpy.array` containing the indexes of the genotypes to remove.

Finds the mean that have a heterozygous genotype for this current marker. Usually used on sexual chromosomes.

`pyGenClean.DupSNPs.duplicated_snps.`**main**(*argString=None*)

The main function of the module..

Here are the steps for duplicated samples:

1.Prints the options.

2.Reads the `map` file to gather marker's position (*readMAP()*).

3.Reads the `tfam` file (*readTFAM()*).

4. Finds the unique markers using the `map` file (*`findUniques()`*).
5. Process the `tped` file. Write a file containing unique markers in `prefix.unique_snps` (`tfam` and `tped`). Keep in memory information about the duplicated markers (`tped`) (*`processTPED()`*).
6. If there are no duplicated markers, the file `prefix.unique_snps` (`tped` and `tfam`) are copied to `prefix.final`.
7. If there are duplicated markers, print a `tped` and `tfam` file containing the duplicated markers (*`printDuplicatedTPEDandTFAM()`*).
8. Computes the frequency of the duplicated markers (using Plink) and read the output file (*`computeFrequency()`*).
9. Computes the concordance and pairwise completion of each of the duplicated markers (*`computeStatistics()`*).
10. Prints the problematic duplicated markers with a file containing the summary of the statistics (completion and pairwise concordance) (*`printProblems()`*).
11. Print the pairwise concordance in a file (matrices) (*`printConcordance()`*).
12. Choose the best duplicated markers using concordance and completion (*`chooseBestSnps()`*).
13. Completes the chosen markers with the remaining duplicated markers (*`createAndCleanTPED()`*).
14. Creates the final `tped` file, containing the unique markers, the chosen duplicated markers that were completed, and the problematic duplicated markers (for further analysis). This set excludes markers that were used for completing the chosen ones (*`createFinalTPEDandTFAM()`*).

`pyGenClean.DupSNPs.duplicated_snps.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The input file prefix (Plink tfile).
`--snp-completion-threshold`	float	The completion threshold to consider a replicate when choosing the best replicates.
`--snp-concordance-threshold`	float	The concordance threshold to consider a replicate when choosing the best replicates.
`--frequency_difference`	float	The maximum difference in frequency between duplicated markers.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.DupSNPs.duplicated_snps.`**`printConcordance`**`(`*concordance*, *prefix*, *tped*, *snps*`)`

Print the concordance.

Parameters

- **`concordance`** (*`dict`*) – the concordance.
- **`prefix`** (*`str`*) – the prefix if the output files.
- **`tped`** (*`numpy.array`*) – a representation of the `tped` of duplicated markers.
- **`snps`** (*`dict`*) – the position of the duplicated markers in the `tped`.

Prints the concordance in a file, in the format of a matrix. For each duplicated markers, the first line (starting with the # signs) contains the name of all the markers in the duplicated markers set. Then a $N \times N$ matrix is printed to file (where N is the number of markers in the duplicated marker list), containing the pairwise concordance.

`pyGenClean.DupSNPs.duplicated_snps.`**`printDuplicatedTPEDandTFAM`**(*tped*, *tfamFileName*, *outPrefix*)

Print the duplicated SNPs TPED and TFAM.

Parameters

- **`tped`** (*`numpy.array`*) – a representation of the `tped` of duplicated markers.
- **`tfamFileName`** (*`str`*) – the name of the original `tfam` file.
- **`outPrefix`** (*`str`*) – the output prefix.

First, it copies the original `tfam` into `outPrefix.duplicated_snps.tfam`. Then, it prints the `tped` of duplicated markers in `outPrefix.duplicated_snps.tped`.

`pyGenClean.DupSNPs.duplicated_snps.`**`printProblems`**(*completion*, *concordance*, *tped*, *snps*, *frequencies*, *prefix*, *diffFreq*)

Print the statistics.

Parameters

- **`completion`** (*`numpy.array`*) – the completion of each duplicated markers.
- **`concordance`** (*`dict`*) – the pairwise concordance between duplicated markers.
- **`tped`** (*`numpy.array`*) – a representation of the `tped` of duplicated markers.
- **`snps`** (*`dict`*) – the positions of the duplicated markers in the `tped`
- **`frequencies`** (*`dict`*) – the frequency of each of the duplicated markers.
- **`prefix`** (*`str`*) – the prefix of the output files.
- **`diffFreq`** (*`float`*) – the frequency difference threshold.

Returns a `set` containing duplicated markers to complete.

Creates a summary file (`prefix.summary`) containing information about duplicated markers: chromosome, position, name, alleles, status, completion percentage, completion number and mean concordance.

The frequency and the minor allele are used to be certain that two duplicated markers are exactly the same marker (and not a tri-allelic one, for example).

For each duplicated markers:

1. Constructs the set of available alleles for the first marker.
2. Constructs the set of available alleles for the second marker.
3. If the two sets are different, but the number of alleles is the same, we try to flip one of the marker. If the two sets are the same, but the number of alleles is 1, we set the status to `homo_flip`. If the markers are heterozygous, we set the status to `flip`.
4. If there is a difference in the number of alleles (one is homozygous, the other, heterozygous), and that there is on allele in common, we set the status to `homo_hetero`. If there are no allele in common, we try to flip one. If the new sets have one allele in common, we set the status to `homo_hetero_flip`.
5. If the sets of available alleles are the same (without flip), we check the frequency and the minor alleles. If the minor allele is different, we set the status to `diff_minor_allele`. If the difference in frequencies is higher than a threshold, we set the status to `diff_frequency`.
6. If all of the above fail, we set the status to `problem`.

Problems are written in the `prefix.problems` file, and contains the following columns: chromosome, position, name and status. This file contains all the markers with a status, as explained above.

`pyGenClean.DupSNPs.duplicated_snps.`**`processTPED`**`(`*uniqueSNPs*, *mapF*, *fileName*, *tfam*, *prefix*`)`

Process the TPED file.

Parameters

- **`uniqueSNPs`** (*`dict`*) – the unique markers.
- **`mapF`** (*`list`*) – a representation of the `map` file.
- **`fileName`** (*`str`*) – the name of the `tped` file.
- **`tfam`** (*`str`*) – the name of the `tfam` file.
- **`prefix`** (*`str`*) – the prefix of all the files.

Returns a tuple with the representation of the `tped` file (`numpy.array`) as first element, and the updated position of the duplicated markers in the `tped` representation.

Copies the `tfam` file into `prefix.unique_snps.tfam`. While reading the `tped` file, creates a new one (`prefix.unique_snps.tped`) containing only unique markers.

`pyGenClean.DupSNPs.duplicated_snps.`**`readMAP`**`(`*fileName*, *prefix*`)`

Reads the MAP file.

Parameters **`fileName`** (*`str`*) – the name of the `map` file.

Returns a list of tuples, representing the `map` file.

While reading the `map` file, it saves a file (`prefix.duplicated_marker_names`) containing the name of the unique duplicated markers.

`pyGenClean.DupSNPs.duplicated_snps.`**`readTFAM`**`(`*fileName*`)`

Reads the TFAM file.

Parameters **`fileName`** (*`str`*) – the name of the `tfam` file.

Returns a representation the `tfam` file (`numpy.array`).

`pyGenClean.DupSNPs.duplicated_snps.`**`runCommand`**`(`*command*`)`

Run the command in Plink.

Parameters **`command`** (*`list`*) – the command to run.

Tries to run a command using `subprocess`.

`pyGenClean.DupSNPs.duplicated_snps.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.DupSamples package

For more information about how to use this module, refer to the *Duplicated Samples Module*.

Module contents

Submodules

pyGenClean.DupSamples.duplicated_samples module

exception pyGenClean.DupSamples.duplicated_samples.**ProgramError**(*msg*)

Bases: exceptions.Exception

An Exception raised in case of a problem.

Parameters msg (*str*) – the message to print to the user before exiting.

pyGenClean.DupSamples.duplicated_samples.**addToTPEDandTFAM**(*tped*, *tfam*, *prefix*, *toAddPrefix*)

Append a tfile to another, creating a new one.

Parameters

- **tped** (numpy.array) – the tped that will be appended to the other one.
- **tfam** (numpy.array) – the tfam that will be appended to the other one.
- **prefix** (*str*) – the prefix of all the files.
- **toAddPrefix** (*str*) – the prefix of the final file.

Here are the steps of this function:

1.Writes the tped into prefix.chosen_samples.tped.

2.Writes the tfam into prefix.chosen_samples.tfam.

3.Copies the previous tfam (toAddPrefix.tfam) into the final tfam (prefix.final.tfam).

4.Append the tfam to the final tfam (prefix.final.tfam).

5.Reads the previous tped (toAddPrefix.tped) and append the new tped to it, writing the final one (prefix.final.tped).

Warning: The tped and tfam variables need to contain at least one sample.

pyGenClean.DupSamples.duplicated_samples.**checkArgs**(*args*)

Checks the arguments and options.

Parameters args (argparse.Namespace) – a argparse.Namespace object containing the options of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the sys.stderr and the program exists with code 1.

pyGenClean.DupSamples.duplicated_samples.**chooseBestDuplicates**(*tped*, *samples*, *oldSamples*, *completion*, *concordance_all*, *prefix*)

Choose the best duplicates according to the completion rate.

Parameters

- **tped** (numpy.array) – the tped containing the duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.

- **completion** (numpy.array) – the completion of each of the duplicated samples.
- **concordance_all** (dict) – the concordance of every duplicated samples.
- **prefix** (str) – the prefix of all the files.

Returns a tuple where the first element is a list of the chosen samples' indexes, the second on is the completion and the last one is the concordance (a map).

These are the steps to find the best duplicated sample:

1. Sort the list of concordances.
2. Sort the list of completions.
3. Choose the best of the concordance and put in a set.
4. Choose the best of the completion and put it in a set.
5. Compute the intersection of the two sets. If there is one sample or more, then randomly choose one sample.
6. If the intersection doesn't contain at least one sample, redo steps 3 and 4, but increase the number of chosen best by one. Redo step 5 and 6 (if required).

The chosen samples are written in `prefix.chosen_samples.info`. The rest are written in `prefix.excluded_samples.info`.

pyGenClean.DupSamples.duplicated_samples.**computeStatistics**(*tped*, *tfam*, *samples*, *oldSamples*, *prefix*)

Computes the completion and concordance of each samples.

Parameters

- **tped** (numpy.array) – the `tped` containing duplicated samples.
- **tfam** (numpy.array) – the `tfam` containing duplicated samples.
- **samples** (dict) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (dict) – the original duplicated sample positions.
- **prefix** (str) – the prefix of all the files.

Returns a tuple containing the completion (`numpy.array`) as first element, and the concordance (`dict`) as last element.

Reads the `tped` file and compute the completion for each duplicated samples and the pairwise concordance between duplicated samples.

Note: The completion and concordance computation excludes a markers if it's on chromosome 24 and if the sample is a female.

Note: A missing genotype is encoded by `0`.

Note: No percentage is computed here, only the numbers. Percentages are computing in other functions: *printStatistics()*, for completion, and *printConcordance()*, for concordance.

Completion

Computes the completion of none zero values (where all genotypes of at least one duplicated sample are no call [*i.e.* `0`]). The completion of sample i (*i.e.* $Comp_i$) is the number of genotypes that have a call divided by the total number of genotypes (the set G_i):

$$Comp_i = \frac{||g \in G_i \text{ where } g \neq 0||}{||G_i||}$$

Note: We consider a genotype as being missing if the sample is a male and if a marker on chromosome 23 or 24 is heterozygous.

Concordance

Computes the pairwise concordance between duplicated samples. For each marker, if both genotypes are not missing, we add one to the total number of compared markers. If both genotypes are the same, we add one to the number of concordant calls. We write the observed genotype difference in the file `prefix.diff`. The concordance between sample i and j (*i.e.* $Concordance_{i,j}$) is the number of genotypes that are equal divided by the total number of genotypes (excluding the no calls):

$$Concordance_{i,j} = \frac{||g \in G_i \cup G_j \text{ where } g_i = g_j \neq 0||}{||g \in G_i \cup G_j \text{ where } g \neq 0||}$$

`pyGenClean.DupSamples.duplicated_samples.`**`createAndCleanTPED`**(*tped*, *tfam*, *samples*, *oldSamples*, *chosenSamples*, *prefix*, *completion*, *completionT*, *concordance*, *concordanceT*)

Complete a TPED for duplicate samples.

Parameters

- **tped** (`numpy.array`) – the `tped` containing the duplicated samples.
- **tfam** (`numpy.array`) – the `tfam` containing the duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the `tped` containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **chosenSamples** (*dict*) – the position of the chosen samples.
- **prefix** (*str*) – the prefix of all the files.
- **completion** (`numpy.array`) – the completion of each of the duplicated samples.
- **completionT** (*float*) – the completion threshold.
- **concordance** (*dict*) – the pairwise concordance of each of the duplicated samples.
- **concordanceT** (*float*) – the concordance threshold.

Using a `tped` containing duplicated samples, it creates a `tped` containing unique samples by completing a chosen sample with the other replicates.

Note: A chosen sample is not completed using bad replicates (those that don't have a concordance or a completion higher than a certain threshold). The bad replicates are written in the file `prefix.not_good_enough`.

pyGenClean.DupSamples.duplicated_samples.**findDuplicates**(*tfam*)

Finds the duplicates in a TFAM.

Parameters **tfam** (*list*) – representation of a tfam file.

Returns two dict, containing unique and duplicated samples position.

pyGenClean.DupSamples.duplicated_samples.**main**(*argString=None*)

Check for duplicated samples in a tfam/tped file.

Parameters **argString** (*list*) – the options

Here are the steps for the duplicated samples step.

1. Prints the options.
2. Reads the tfam file (*readTFAM()*).
3. Separate the duplicated samples from the unique samples (*findDuplicates()*).
4. Writes the unique samples into a file named prefix.unique_samples.tfam (*printUniqueTFAM()*).
5. Reads the tped file and write into prefix.unique_samples.tped the pedigree file for the unique samples (*processTPED()*). Saves in memory the pedigree for the duplicated samples. Updates the indexes of the duplicated samples.
6. If there are no duplicated samples, simply copies the files prefix.unique_samples (tped and tfam) to prefix.final.tfam and prefix..final.tped, respectively.
7. Computes the completion (for each of the duplicated samples) and the concordance of each sample pairs (*computeStatistics()*).
8. Prints statistics (concordance and completion) (*printStatistics()*).
9. We print the concordance matrix for each duplicated samples (*printConcordance()*).
10. We print the tped and the tfam file for the duplicated samples (prefix.duplicated_samples) (*printDuplicatedTPEDandTFAM()*).
11. Choose the best of each duplicates (to keep and to complete) according to completion and concordance (*chooseBestDuplicates()*).
12. Creates a unique tped and tfam from the duplicated samples by completing the best chosen one with the other samples (*createAndCleanTPED()*).
13. Merge the two tfiles together (prefix.unique_samples and prefix.chosen_samples) to create the final dataset (prefix.final) (*addToTPEDandTFAM()*).

pyGenClean.DupSamples.duplicated_samples.**parseArgs**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list*) – the options

Returns A argparse.Namespace object created by the argparse module. It contains the values of the different options.

Options	Type	Description
--tfile	string	The input file prefix (of type tfile).
--sample-completion-threshold	float	The completion threshold.
--sample-concordance-threshold	float	The concordance threshold.
--out	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.DupSamples.duplicated_samples.`**`printConcordance`**`(`*concordance*, *prefix*`)`

Print the concordance.

Parameters

- **`concordance`** (`dict`) – the concordance of each sample.
- **`prefix`** (`str`) – the prefix of all the files.

Returns the concordance percentage (dict)

The concordance is the number of genotypes that are equal when comparing a duplicated samples with another one, divided by the total number of genotypes (excluding genotypes that are no call [*i.e.* `0`]). If a duplicated sample has 100% of no calls, the concordance will be zero.

The file `prefix.concordance` will contain $N \times N$ matrices for each set of duplicated samples.

`pyGenClean.DupSamples.duplicated_samples.`**`printDuplicatedTPEDandTFAM`**`(`*tped*, *tfam*, *samples*, *oldSamples*, *prefix*`)`

Print the TPED and TFAM of the duplicated samples.

Parameters

- **`tped`** (`numpy.array`) – the `tped` containing duplicated samples.
- **`tfam`** (`numpy.array`) – the `tfam` containing duplicated samples.
- **`samples`** (`dict`) – the updated position of the samples in the tped containing only duplicated samples.
- **`oldSamples`** (`dict`) – the original duplicated sample positions.
- **`prefix`** (`str`) – the prefix of all the files.

The `tped` and `tfam` files are written in `prefix.duplicated_samples.tped` and `prefix.duplicated_samples.tfam`, respectively.

`pyGenClean.DupSamples.duplicated_samples.`**`printStatistics`**`(`*completion*, *concordance*, *tpedSamples*, *oldSamples*, *prefix*`)`

Print the statistics in a file.

Parameters

- **`completion`** (`numpy.array`) – the completion of each duplicated samples.
- **`concordance`** (`dict`) – the concordance of each duplicated samples.
- **`tpedSamples`** (`dict`) – the updated position of the samples in the tped containing only duplicated samples.
- **`oldSamples`** (`dict`) – the original duplicated sample positions.
- **`prefix`** (`str`) – the prefix of all the files.

Returns the completion for each duplicated samples, as a `numpy.array`.

Prints the statistics (completion of each samples and pairwise concordance between duplicated samples) in a file (`prefix.summary`).

`pyGenClean.DupSamples.duplicated_samples.`**`printUniqueTFAM`**`(`*tfam*, *samples*, *prefix*`)`

Prints a new TFAM with only unique samples.

Parameters

- **`tfam`** (*`list`*) – a representation of a TFAM file.
- **`samples`** (*`dict`*) – the position of the samples
- **`prefix`** (*`str`*) – the prefix of the output file name

`pyGenClean.DupSamples.duplicated_samples.`**`processTPED`**`(`*uniqueSamples*, *duplicatedSamples*, *fileName*, *prefix*`)`

Process the TPED file.

Parameters

- **`uniqueSamples`** (*`dict`*) – the position of unique samples.
- **`duplicatedSamples`** (*`collections.defaultdict`*) – the position of duplicated samples.
- **`fileName`** (*`str`*) – the name of the file.
- **`prefix`** (*`str`*) – the prefix of all the files.

Returns a tuple containing the `tped` (`numpy.array`) as first element, and the updated positions of the duplicated samples (`dict`)

Reads the entire `tped` and prints another one containing only unique samples (`prefix.unique_samples.tped`). It then creates a `numpy.array` containing the duplicated samples.

`pyGenClean.DupSamples.duplicated_samples.`**`readTFAM`**`(`*fileName*`)`

Reads the TFAM file.

Parameters **`fileName`** (*`str`*) – the name of the `tfam` file.

Returns a list of tuples, representing the `tfam` file.

`pyGenClean.DupSamples.duplicated_samples.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.Ethnicity package

For more information about how to use this module, refer to the *Ethnicity Module*.

Module contents

Submodules

pyGenClean.Ethnicity.check_ethnicity module

class `pyGenClean.Ethnicity.check_ethnicity.`**`Dummy`**

Bases: `object`

exception `pyGenClean.Ethnicity.check_ethnicity.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (`str`) – the message to print to the user before exiting.

`pyGenClean.Ethnicity.check_ethnicity.`**`allFileExists`**(*fileList*)

Check that all file exists.

Parameters **`fileList`** (`list`) – the list of file to check.

Check if all the files in `fileList` exists.

`pyGenClean.Ethnicity.check_ethnicity.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.Ethnicity.check_ethnicity.`**`combinePlinkBinaryFiles`**(*prefixes*, *outPrefix*)

Combine Plink binary files.

Parameters

- **`prefixes`** (`list`) – a list of the prefix of the files that need to be combined.
- **`outPrefix`** (`str`) – the prefix of the output file (the combined file).

It uses Plink to merge a list of binary files (which is a list of prefixes (strings)), and create the final data set which as `outPrefix` as the prefix.

`pyGenClean.Ethnicity.check_ethnicity.`**`computeFrequency`**(*prefix*, *outPrefix*)

Compute the frequency using Plink.

Parameters

- **`prefix`** (`str`) – the prefix of the file binary file for which we need to compute frequencies.
- **`outPrefix`** (`str`) – the prefix of the output files.

Uses Plink to compute the frequency of all the markers in the `prefix` binary file.

`pyGenClean.Ethnicity.check_ethnicity.`**`compute_eigenvalues`**(*in_prefix*, *out_prefix*)

Computes the Eigenvalues using smartpca from Eigensoft.

Parameters

- **`in_prefix`** (`str`) – the prefix of the input files.
- **`out_prefix`** (`str`) – the prefix of the output files.

Creates a “parameter file” used by smartpca and runs it.

`pyGenClean.Ethnicity.check_ethnicity.`**`createMDSFile`**(*nb_components*, *inPrefix*, *outPrefix*, *genomeFileName*)

Creates a MDS file using Plink.

Parameters

- **`nb_components`** (`int`) – the number of component.
- **`inPrefix`** (`str`) – the prefix of the input file.
- **`outPrefix`** (`str`) – the prefix of the output file.

- **genomeFileName** (*str*) – the name of the genome file.

Using Plink, computes the MDS values for each individual using the inPrefix, genomeFileName and the number of components. The results are save using the outPrefix prefix.

pyGenClean.Ethnicity.check_ethnicity.**createPopulationFile**(*inputFiles*, *labels*, *outputFileName*)

Creates a population file.

Parameters

- **inputFiles** (*list*) – the list of input files.
- **labels** (*list*) – the list of labels (corresponding to the input files).
- **outputFileName** (*str*) – the name of the output file.

The inputFiles is in reality a list of tfam files composed of samples. For each of those tfam files, there is a label associated with it (representing the name of the population).

The output file consists of one row per sample, with the following three columns: the family ID, the individual ID and the population of each sample.

pyGenClean.Ethnicity.check_ethnicity.**create_scree_plot**(*in_filename*, *out_filename*, *plot_title*)

Creates a scree plot using smartpca results.

Parameters

- **in_filename** (*str*) – the name of the input file.
- **out_filename** (*str*) – the name of the output file.
- **plot_title** (*str*) – the title of the scree plot.

pyGenClean.Ethnicity.check_ethnicity.**excludeSNPs**(*inPrefix*, *outPrefix*, *exclusionFileName*)

Exclude some SNPs using Plink.

Parameters

- **inPrefix** (*str*) – the prefix of the input file.
- **outPrefix** (*str*) – the prefix of the output file.
- **exclusionFileName** (*str*) – the name of the file containing the markers to be excluded.

Using Plink, exclude a list of markers from inPrefix, and saves the results in outPrefix. The list of markers are in exclusionFileName.

pyGenClean.Ethnicity.check_ethnicity.**extractSNPs**(*snpToExtractFileNames*, *referencePrefixes*, *popNames*, *outPrefix*, *runSGE*, *options*)

Extract a list of SNPs using Plink.

Parameters

- **snpToExtractFileNames** (*list*) – the name of the files which contains the markers to extract from the original data set.
- **referencePrefixes** (*list*) – a list containing the three reference population prefixes (the original data sets).
- **popNames** (*list*) – a list containing the three reference population names.
- **outPrefix** (*str*) – the prefix of the output file.

- **runSGE** (*boolean*) – Whether using SGE or not.
- **options** (*argparse.Namespace*) – the options.

Using Plink, extract a set of markers from a list of prefixes.

`pyGenClean.Ethnicity.check_ethnicity.`**`findFlippedSNPs`**(*frqFile1*, *frqFile2*, *outPrefix*)

Find flipped SNPs and flip them in the data.

Parameters

- **frqFile1** (*str*) – the name of the first frequency file.
- **frqFile2** (*str*) – the name of the second frequency file.
- **outPrefix** (*str*) – the prefix of the output files.

By reading two frequency files (`frqFile1` and `frqFile2`), it finds a list of markers that need to be flipped so that the first file becomes comparable with the second one. Also finds marker that need to be removed.

A marker needs to be flipped in one of the two data set if the two markers are not comparable (same minor allele), but become comparable if we flip one of them.

A marker will be removed if it is all homozygous in at least one data set. It will also be removed if it's impossible to determine the phase of the marker (*e.g.* if the two alleles are `A` and `T` or `C` and `G`).

`pyGenClean.Ethnicity.check_ethnicity.`**`findOverlappingSNPsWithReference`**(*prefix*, *referencePrefixes*, *referencePopulations*, *outPrefix*)

Find the overlapping SNPs in 4 different data sets.

Parameters

- **prefix** (*str*) – the prefix of all the files.
- **referencePrefixes** (*list*) – the prefix of the reference population files.
- **referencePopulations** (*list*) – the name of the reference population (same order as `referencePrefixes`)
- **outPrefix** (*str*) – the prefix of the output files.

It starts by reading the `bim` file of the source data set (`prefix.bim`). It finds all the markers (excluding the duplicated ones). Then it reads all of the reference population `bim` files (`referencePrefixes.bim`) and find all the markers that were found in the source data set.

It creates three output files:

- `outPrefix.ref_snp_to_extract`: the name of the markers that needs to be extracted from the three reference panels.
- `outPrefix.source_snp_to_extract`: the name of the markers that needs to be extracted from the source panel.

- `outPrefix.update_names`: a file (readable by Plink) that will help in changing the names of the selected markers in the reference panels, so that they become comparable with the source panel.

`pyGenClean.Ethnicity.check_ethnicity.`**`find_the_outliers`**(*mds_file_name*, *population_file_name*, *ref_pop_name*, *multiplier*, *out_prefix*)

Finds the outliers of a given population.

Parameters

- **`mds_file_name`** (*`str`*) – the name of the `mds` file.
- **`population_file_name`** (*`str`*) – the name of the population file.
- **`ref_pop_name`** (*`str`*) – the name of the reference population for which to find outliers from.
- **`multiplier`** (*`float`*) – the multiplier of the cluster standard deviation to modify the strictness of the outlier removal procedure.
- **`out_prefix`** (*`str`*) – the prefix of the output file.

Uses the *`pyGenClean.Ethnicity.find_outliers`* modules to find outliers. It requires the `mds` file created by *`createMDSFile()`* and the population file created by *`createPopulationFile()`*.

`pyGenClean.Ethnicity.check_ethnicity.`**`flipSNPs`**(*inPrefix*, *outPrefix*, *flipFileName*)

Flip SNPs using Plink.

Parameters

- **`inPrefix`** (*`str`*) – the prefix of the input file.
- **`outPrefix`** (*`str`*) – the prefix of the output file.
- **`flipFileName`** (*`str`*) – the name of the file containing the markers to flip.

Using Plink, flip a set of markers in `inPrefix`, and saves the results in `outPrefix`. The list of markers to be flipped is in `flipFileName`.

`pyGenClean.Ethnicity.check_ethnicity.`**`main`**(*argString=None*)

The main function.

Parameters **`argString`** (*`list`*) – the options.

These are the steps of this module:

1. Prints the options.
2. Finds the overlapping markers between the three reference panels and the source panel (*`findOverlappingSNPsWithReference()`*).
3. Extract the required markers from all the data sets (*`extractSNPs()`*).
4. Renames the reference panel's marker names to that they are the same as the source panel (for all populations) (*`renameSNPs()`*).
5. Combines the three reference panels together (*`combinePlinkBinaryFiles()`*).
6. Compute the frequency of all the markers from the reference and the source panels (*`computeFrequency()`*).
7. Finds the markers to flip in the reference panel (when compared to the source panel) (*`findFlippedSNPs()`*).
8. Excludes the unflippable markers from the reference and the source panels (*`excludeSNPs()`*).

9. Flips the markers that need flipping in their reference panel (`flipSNPs()`).
10. Combines the reference and the source panels (`combinePlinkBinaryFiles()`).
11. Runs part of `pyGenClean.RelatedSamples.find_related_samples` on the combined data set (`runRelatedness()`).
12. Creates the `mds` file from the combined data set and the result of previous step (`createMDSFile()`).
13. Creates the population file (`createPopulationFile()`).
14. Plots the `mds` values (`plotMDS()`).
15. Finds the outliers of a given reference population (`find_the_outliers()`).
16. If required, computes the Eigenvalues using smartpca (`compute_eigenvalues()`).
17. If required, creates a scree plot from smartpca resutls (`create_scree_plot()`).

`pyGenClean.Ethnicity.check_ethnicity.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **`argString`** (`list`) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--skip-ref-pops`	bool	Perform the MDS computation, but skip the three reference panels.
`--ceu-bfile`	string	The input file prefix for the CEU population (Plink binary file).
`--yri-bfile`	string	The input file prefix for the YRI population (Plink binary file).
`--jpt-chb-bfile`	string	The input file prefix for the JPT-CHB population (Plink binary file).
`--min-nb-snp`	int	The minimum number of markers needed to compute IBS.
`--indep-pairwise`	string	Three numbers: window size, window shift and the r2 threshold.
`--maf`	string	Restrict to SNPs with MAF >= threshold.
`--sge`	bool	Use SGE for parallelization.
`--sge-walltime`	int	The time limit (for clusters).
`--sge-nodes`	int int	Two INTs (number of nodes and number of processor per nodes).
`--ibs-sge-walltime`	int	The time limit (for clusters) (for IBS)
`--ibs-sge-nodes`	int int	Two INTs (number of nodes and number of processor per nodes) (for IBS).
`--line-per-file-for-sge`	int	The number of line per file for SGE task array.
`--nb-components`	int	The number of component to compute.
`--outliers-of`	string	Finds the ouliers of this population.
`--multiplier`	float	To find the outliers, we look for more than x times the cluster standard deviation.
`--xaxis`	string	The component to use for the X axis.
`--yaxis`	string	The component to use for the Y axis.
`--format`	string	The output file format.
`--title`	string	The title of the MDS plot.
`--xlabel`	string	The label of the X axis.
`--ylabel`	string	The label of the Y axis.
`--create-scree-plot`	bool	Computes Eigenvalues and creates a scree plot.
`--scree-plot-title`	string	The main title of the scree plot
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.Ethnicity.check_ethnicity.`**`plotMDS`**`(`*inputFileName*, *outPrefix*, *populationFileName*, *options*`)`

Plots the MDS value.

Parameters

- **`inputFileName`** (`str`) – the name of the `mds` file.
- **`outPrefix`** (`str`) – the prefix of the output files.
- **`populationFileName`** (`str`) – the name of the population file.
- **`options`** (`argparse.Namespace`) – the options

Plots the `mds` value according to the `inputFileName` file (`mds`) and the `populationFileName` (the population file).

`pyGenClean.Ethnicity.check_ethnicity.`**`renameSNPs`**`(`*inPrefix*, *updateFileName*, *outPrefix*`)`

Updates the name of the SNPs using Plink.

Parameters

- **`inPrefix`** (`str`) – the prefix of the input file.
- **`updateFileName`** (`str`) – the name of the file containing the updated marker names.
- **`outPrefix`** (`str`) – the prefix of the output file.

Using Plink, changes the name of the markers in `inPrefix` using `updateFileName`. It saves the results in `outPrefix`.

`pyGenClean.Ethnicity.check_ethnicity.`**`runCommand`**`(`*command*`)`

Run a command.

Parameters **`command`** (`list`) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable `command` should be a list of strings (no other type).

`pyGenClean.Ethnicity.check_ethnicity.`**`runRelatedness`**`(`*inputPrefix*, *outPrefix*, *options*`)`

Run the relatedness step of the data clean up.

Parameters

- **`inputPrefix`** (`str`) – the prefix of the input file.
- **`outPrefix`** (`str`) – the prefix of the output file.
- **`options`** (`argparse.Namespace`) – the options

Returns the prefix of the new bfile.

Runs `pyGenClean.RelatedSamples.find_related_samples` using the `inputPrefix` files and `options` options, and saves the results using the `outPrefix` prefix.

`pyGenClean.Ethnicity.check_ethnicity.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.Ethnicity.find_outliers module

exception `pyGenClean.Ethnicity.find_outliers.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.Ethnicity.find_outliers.`**`add_custom_options`**(*parser*)

Adds custom options to a parser.

Parameters **`parser`** (*`argparse.ArgumentParser`*) – the parser to which to add options.

`pyGenClean.Ethnicity.find_outliers.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – a `argparse.Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.Ethnicity.find_outliers.`**`find_outliers`**(*mds*, *centers*, *center_info*, *ref_pop*, *options*)

Finds the outliers for a given population.

Parameters

- **`mds`** (*`numpy.recarray`*) – the `mds` information about each samples.
- **`centers`** (*`numpy.array`*) – the centers of the three reference population clusters.
- **`center_info`** (*`dict`*) – the label of the three reference population clusters.
- **`ref_pop`** (*`str`*) – the reference population for which we need the outliers from.
- **`options`** (*`argparse.Namespace`*) – the options

Returns a `set` of outliers from the `ref_pop` population.

Perform a `KMeans` classification using the three centers from the three reference population cluster.

Samples are outliers of the required reference population (`ref_pop`) if:

- the sample is part of another reference population cluster;
- the sample is an outlier of the desired reference population (`ref_pop`).

A sample is an outlier of a given cluster C_j if the distance between this sample and the center of the cluster C_j (O_j) is bigger than a constant times the cluster's standard deviation σ_j.

$$\sigma_j = \sqrt{\frac{\sum d(s_i, O_j)^2}{||C_j|| - 1}}$$

where $||C_j||$ is the number of samples in the cluster C_j, and $d(s_i, O_j)$ is the distance between the sample s_i and the center O_j of the cluster C_j.

$$d(s_i, O_j) = \sqrt{(x_{O_j} - x_{s_i})^2 + (y_{O_j} - y_{s_i})^2}$$

Using a constant equals to one ensure we remove 100% of the outliers from the cluster. Using a constant of 1.6 or 1.9 ensures we remove 99% and 95% of outliers, respectively (an error rate of 1% and 5%, respectively).

`pyGenClean.Ethnicity.find_outliers.`**`find_ref_centers`**(*mds*)

Finds the center of the three reference clusters.

Parameters **`mds`** (*`numpy.recarray`*) – the `mds` information about each samples.

Returns a tuple with a `numpy.array` containing the centers of the three reference population cluster as first element, and a `dict` containing the label of each of the three reference population clusters.

First, we extract the `mds` values of each of the three reference populations. The, we compute the center of each of those clusters by computing the means.

$$\text{Cluster}_{\text{pop}} = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}\right)$$

`pyGenClean.Ethnicity.find_outliers.`**`main`**(*argString=None*)

The main function.

Parameters **`argString`** (*`list of strings`*) – the options.

These are the steps of the modules:

1. Prints the options.
2. Reads the population file (*`read_population_file()`*).
3. Reads the `mds` file (*`read_mds_file()`*).
4. Computes the three reference population clusters' centers (*`find_ref_centers()`*).
5. Computes three clusters according to the reference population clusters' centers, and finds the outliers of a given reference population (*`find_outliers()`*). This steps also produce three different plots.
6. Writes outliers in a file (`prefix.outliers`).

`pyGenClean.Ethnicity.find_outliers.`**`overwrite_tex`**(*tex_fn*, *nb_outliers*, *script_options*)

Overwrites the TeX summary file with new values.

Parameters

- **`tex_fn`** (*`str`*) – the name of the TeX summary file to overwrite.
- **`nb_outliers`** (*`int`*) – the number of outliers.
- **`script_options`** – the script options.

`pyGenClean.Ethnicity.find_outliers.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--mds`	string	The MDS file from Plink.
`--population-file`	string	A population file from `pyGenClean.Ethnicity.check_ethnicity` module.
`--format`	string	The output file format (png, ps, or pdf.
`--outliers-of`	string	Finds the outliers of this population.
`--multiplier`	float	To find the outliers, we look for more than x times the cluster standard deviation.
`--xaxis`	string	The component to use for the X axis.
`--yaxis`	string	The component to use for the Y axis.
`--format`	string	The output file format (png, ps, or pdf formats are available).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.Ethnicity.find_outliers.`**`read_mds_file`**(*file_name*, *c1*, *c2*, *pops*)

Reads a MDS file.

Parameters

- **`file_name`** (`str`) – the name of the `mds` file.
- **`c1`** (`str`) – the first component to read (x axis).
- **`c2`** (`str`) – the second component to read (y axis).
- **`pops`** (`dict`) – the population of each sample.

Returns a `numpy.recarray` (one sample per line) with the information about the family ID, the individual ID, the first component to extract, the second component to extract and the population.

The `mds` file is the result of Plink (as produced by the `pyGenClean.Ethnicity.check_ethnicity` module).

`pyGenClean.Ethnicity.find_outliers.`**`read_population_file`**(*file_name*)

Reads the population file.

Parameters **`file_name`** (`str`) – the name of the population file.

Returns a `dict` containing the population for each of the samples.

The population file should contain three columns:

1.The family ID.

2.The individual ID.

3.The population of the file (one of `CEU`, `YRI`, `JPT-CHB` or `SOURCE`).

The outliers are from the `SOURCE` population, when compared to one of the three reference population (`CEU`, `YRI` or `JPT-CHB`).

`pyGenClean.Ethnicity.find_outliers.`**`safe_main`**()

A safe version of the main function (that catches ProgramError).

pyGenClean.Ethnicity.plot_eigenvalues module

exception `pyGenClean.Ethnicity.plot_eigenvalues.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.Ethnicity.plot_eigenvalues.`**`add_custom_options`**(*parser*)

Adds custom options to a parser.

Parameters **`parser`** (*`argparse.ArgumentParser`*) – the parser to which the options will be added.

`pyGenClean.Ethnicity.plot_eigenvalues.`**`check_args`**(*args*)

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options and arguments of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exits with error code 1.

`pyGenClean.Ethnicity.plot_eigenvalues.`**`create_scree_plot`**(*data*, *o_filename*, *options*)

Creates the scree plot.

Parameters

- **`data`** (*`numpy.ndarray`*) – the eigenvalues.
- **`o_filename`** (*`str`*) – the name of the output files.
- **`options`** (*`argparse.Namespace`*) – the options.

`pyGenClean.Ethnicity.plot_eigenvalues.`**`main`**(*argString=None*)

The main function.

The purpose of this module is to plot Eigenvectors provided by the Eigensoft software.

Here are the steps of this module:

1.Reads the Eigenvector (*`read_eigenvalues()`*).

2.Plots the Scree Plot (*`create_scree_plot()`*).

`pyGenClean.Ethnicity.plot_eigenvalues.`**`parse_args`**(*argString=None*)

Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--evec`	string	The EVEC file from EIGENSOFT
`--scree-plot-title`	string	The main title of the scree plot
`--out`	string	The name of the output file

Note: No option check is done here (except for the one automatically done by `argparse`). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.Ethnicity.plot_eigenvalues.`**`read_eigenvalues`**(*i_filename*)

Reads the eigenvalues from EIGENSOFT results.

Parameters **`i_filename`** (*`str`*) – the name of the input file.

Returns a `numpy.ndarray` array containing the eigenvalues.

`pyGenClean.Ethnicity.plot_eigenvalues.`**`safe_main`**`()`
A safe version of the main function (that catches ProgramError).

pyGenClean.FlagHW package

For more information about how to use this module, refer to the *Hardy Weinberg Equilibrium Module*.

Module contents

Submodules

pyGenClean.FlagHW.flag_hw module

class `pyGenClean.FlagHW.flag_hw.`**`Dummy`**
Bases: `object`

exception `pyGenClean.FlagHW.flag_hw.`**`ProgramError`**`(`*msg*`)`
Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (`str`) – the message to print to the user before exiting.

`pyGenClean.FlagHW.flag_hw.`**`checkArgs`**`(`*args*`)`
Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – a `argparse.Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.FlagHW.flag_hw.`**`compareBIMfiles`**`(`*beforeFileName*, *afterFileName*, *outputFileName*`)`
Compare two BIM files for differences.

Parameters

- **`beforeFileName`** (`str`) – the name of the file before modification.
- **`afterFileName`** (`str`) – the name of the file after modification.
- **`outputFileName`** (`str`) – the name of the output file (containing the differences between the `before` and the `after` files.

Returns the number of differences between the two files.

The `bim` files contain the list of markers in a given dataset. The `before` file should have more markers than the `after` file. The `after` file should be a subset of the markers in the `before` file.

`pyGenClean.FlagHW.flag_hw.`**`computeHWE`**`(`*prefix*, *threshold*, *outPrefix*`)`
Compute the Hardy Weinberg test using Plink.

Parameters

- **`prefix`** (`str`) – the prefix of all the files.
- **`threshold`** (`str`) – the Hardy Weinberg threshold.

- **outPrefix** (*str*) – the prefix of the output file.

Uses Plink to exclude markers that failed the Hardy-Weinberg test at a specified significance threshold.

`pyGenClean.FlagHW.flag_hw.`**`computeNumberOfMarkers`**(*inputFileName*)

Count the number of marker (line) in a BIM file.

Parameters **inputFileName** (*str*) – the name of the `bim` file.

Returns the number of marker in the `bim` file.

`pyGenClean.FlagHW.flag_hw.`**`main`**(*argString=None*)

The main function.

Parameters **argString** (*list*) – the options.

These are the steps performed by this module:

1.Prints the options of the module.

2.Computes the number of markers in the input file (*computeNumberOfMarkers()*).

3.If there are no markers, the module stops.

4.Computes the Bonferroni therhold ($0.05/\mathrm{nbMarkers}$).

5.Runs Plink to find failed markers with the Bonferroni threshold.

6.Runs Plink to find failed markers with the default threshold.

7.Compares the `bim` files for the Bonferroni threshold.

8.Compares the `bim` files for the default threshold.

9.Computes the "in between" marker list, which is the markers from the default threshold and the Bonferroni threshold.

`pyGenClean.FlagHW.flag_hw.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (binary Plink file).
`--hwe`	float	The Hardy-Weinberg equilibrium threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *checkArgs()*).

`pyGenClean.FlagHW.flag_hw.`**`runCommand`**(*command*)

Run a command.

Parameters **command** (*list*) – the command to run.

Tries to run a command. If it fails, raise a *ProgramError*. This function uses the `subprocess` module.

Warning: The variable `command` should be a list of stings (no other type).

`pyGenClean.FlagHW.flag_hw.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.FlagMAF package

For more information about how to use this module, refer to the *Minor Allele Frequency of Zero Module*.

Module contents

Submodules

pyGenClean.FlagMAF.flag_maf_zero module

exception `pyGenClean.FlagMAF.flag_maf_zero.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.FlagMAF.flag_maf_zero.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – a `argparse.Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.FlagMAF.flag_maf_zero.`**`computeFrequency`**`(`*options*`)`

Compute the frequency of the SNPs.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

`pyGenClean.FlagMAF.flag_maf_zero.`**`findSnpWithMaf0`**`(`*freqFileName*, *prefix*`)`

Finds SNPs with MAF of 0 and put them in a file.

Parameters

- **`freqFileName`** (*`str`*) – the name of the frequency file.
- **`prefix`** (*`str`*) – the prefix of all the files.

Reads a frequency file from Plink, and find markers with a minor allele frequency of zero.

`pyGenClean.FlagMAF.flag_maf_zero.`**`main`**`(`*argString=None*`)`

The main function.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1.Prints the options.

2.Computes the frequencies using Plinl (*`computeFrequency()`*).

3.Finds markers with MAF of 0, and saves them in a file (*`findSnpWithMaf0()`*).

`pyGenClean.FlagMAF.flag_maf_zero.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.FlagMAF.flag_maf_zero.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.HeteroHap package

For more information about how to use this module, refer to the *Heterozygous Haploid Module*.

Module contents

Submodules

pyGenClean.HeteroHap.remove_heterozygous_haploid module

exception `pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – a an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`main`**`(`*argString=None*`)`

The main function of this module.

Parameters **`argString`** (*`list`*) – the options.

`pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`runPlink`**`(`*options*`)`
Sets heterozygous haploid markers to missing Plink.

Parameters **`options`** (`argparse.Namespace`) – the options.

`pyGenClean.HeteroHap.remove_heterozygous_haploid.`**`safe_main`**`()`
A safe version of the main function (that catches ProgramError).

pyGenClean.LaTeX package

Submodules

pyGenClean.LaTeX.auto_report module

`pyGenClean.LaTeX.auto_report.`**`create_report`**`(`*outdirname*, *report_filename*, ***kwargs*`)`
Creates a LaTeX report.

Parameters

- **`report_filename`** (`str`) – the name of the file.
- **`outdirname`** (`str`) – the name of the output directory.

pyGenClean.LaTeX.merge_reports module

exception `pyGenClean.LaTeX.merge_reports.`**`ProgramError`**`(`*msg*`)`
Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (`str`) – the message to print to the user before exiting.

`pyGenClean.LaTeX.merge_reports.`**`add_custom_options`**`(`*parser*`)`
Adds custom options to a parser.

Parameters **`parser`** (`argparse.ArgumentParser`) – the parser to which to add options.

`pyGenClean.LaTeX.merge_reports.`**`checkArgs`**`(`*args*`)`
Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.LaTeX.merge_reports.`**`copy_initial_files`**`(`*filename*, *out_dir*`)`
Copy the initial_files file to the final directory.

Parameters

- **filename** – the name of the file.
- **out_dir** (*str*) – the name of the output directory

`pyGenClean.LaTeX.merge_reports.`**`generate_report`**(*out_dir*, *latex_summaries*, *nb_markers*, *nb_samples*, *options*)

Generates the report.

Parameters

- **out_dir** (*str*) – the output directory.
- **latex_summaries** (*list*) – the list of LaTeX summaries.
- **nb_markers** (*str*) – the final number of markers.
- **nb_samples** (*str*) – the final number of samples.
- **options** (*argparse.Namespace*) – the list of options.

`pyGenClean.LaTeX.merge_reports.`**`get_final_numbers`**(*filename*, *out_dir*)

Copy the final_files file and get the number of markers and samples.

Parameters

- **filename** (*str*) – the name of the file.
- **out_dir** (*str*) – the output directory.

Returns the final number of markers and samples

Return type tuple

`pyGenClean.LaTeX.merge_reports.`**`get_summary_files`**(*dirnames*)

Gets the TeX summary files for each test.

Parameters **dirnames** (*list*) – the list of directories to merge data from.

Returns a list of summary file names.

Return type list

`pyGenClean.LaTeX.merge_reports.`**`main`**(*argString=None*)

The main function of this module.

Parameters **argString** (*list of strings*) – the options.

`pyGenClean.LaTeX.merge_reports.`**`merge_required_files`**(*dirnames*, *out_dir*)

Merges the required files from each of the directories.

Parameters

- **dirnames** (*list*) – the list of directories to merge data from.
- **out_dir** (*str*) – the name of the output directory.

`pyGenClean.LaTeX.merge_reports.`**`order_qc_dir`**(*dirnames*)

Order the QC directory names according to their date.

Parameters **dirnames** (*list*) – the list of directories to merge data from.

Returns the sorted list of directories

Return type list

`pyGenClean.LaTeX.merge_reports.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--report-author`	String	The current project number.
`--report-number`	String	The current project author.
`--report-background`	String	Text of file containing the background section of the report.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.LaTeX.merge_reports.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.LaTeX.utils module

`pyGenClean.LaTeX.utils.`**`bib_entry`**`(`*`**kwargs`*`)`

Creates a bibliography entry.

Parameters

- **`name`** (*`str`*) – the name of the entry.
- **`authors`** (*`str`*) – the authors.
- **`title`** (*`str`*) – the title.
- **`journal`** (*`str`*) – the journal.
- **`year`** (*`str`*) – the year.
- **`volume`** (*`str`*) – the volume.
- **`number`** (*`str`*) – the number.
- **`pages`** (*`str`*) – the pages.

Returns a bib entry.

`pyGenClean.LaTeX.utils.`**`format_numbers`**`(`*number*, *prefix=''*`)`

Formats number in the scientific notation for LaTeX.

Parameters

- **`number`** (*`str`*) – the number to format.
- **`prefix`** (*`str`*) – a prefix to add before the number (e.g. “p < ”).

Returns a string containing the scientific notation of the number.

Return type str

`pyGenClean.LaTeX.utils.`**`inline_math`**`(`*text*`)`

Creates an inline mathematical formula.

`pyGenClean.LaTeX.utils.`**`item`**`(`*text*`)`

Returns an LaTeX item (for enumerate or itemize).

Parameters **`text`** (*`str`*) – the text.

Returns a LaTeX item.

pyGenClean.LaTeX.utils.**sanitize_fig_name**(*name*)

Sanitizes the name of a file (for including graphics in LaTeX).

Parameters **name** (*str*) – the name of the file to sanitize.

Returns the sanitized name.

For example, if the name of the graphic file is test.1.png, the sanitized version of it for LaTeX is {test.1}.png.

pyGenClean.LaTeX.utils.**sanitize_tex**(*original_text*)

Sanitize TeX text.

Parameters **original_text** (*str*) – the text to sanitize for LaTeX.

Returns the sanitize text.

Text is sanitized by following these steps:

1.Replaces \` by ``\textbackslash

2.Escapes certain characters (such as $, %, _, }, {, & and #) by adding a backslash (*e.g.* from & to \&).

3.Replaces special characters such as ~ by the LaTeX equivalent (*e.g.* from ~ to $\sim$).

pyGenClean.LaTeX.utils.**section**(*name*)

Creates a new section in LaTeX.

Parameters **name** (*str*) – the name of the new section.

Returns a LaTeX string containing a new section.

Return type str

pyGenClean.LaTeX.utils.**subsection**(*name*)

Creates a new section in LaTeX.

Parameters **name** (*str*) – the name of the new section.

Returns a LaTeX string containing a new section.

Return type str

pyGenClean.LaTeX.utils.**textbf**(*text*)

Returns bold font.

Parameters **text** (*str*) – the text.

Returns an bold font.

pyGenClean.LaTeX.utils.**textit**(*text*)

Returns italic font.

Parameters **text** (*str*) – the text.

Returns an italic font.

pyGenClean.LaTeX.utils.**texttt**(*text*)

Returns type writer font.

Parameters **text** (*str*) – the text.

Returns a type writer font.

pyGenClean.LaTeX.utils.**wrap_lines**(*content*, *length=80*)

Wraps long lines to a maximum length of 80.

Parameters

- **content** (*str*) – the content to wrap.
- **legnth** – the maximum length to wrap the content.

Returns a string containing the wrapped content.

Return type str

Module contents

pyGenClean.MarkerMissingness package

For more information about how to use this module, refer to the *Marker Missingness Module*.

Module contents

Submodules

pyGenClean.MarkerMissingness.snp_missingness module

exception `pyGenClean.MarkerMissingness.snp_missingness.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **msg** (*str*) – the message to print to the user before exiting.

`pyGenClean.MarkerMissingness.snp_missingness.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.MarkerMissingness.snp_missingness.`**`compareBIM`**(*args*)

Compare two BIM file.

Parameters **args** (*argparse.Namespace*) – the options.

Creates a *Dummy* object to mimic an `argparse.Namespace` class containing the options for the *pyGenClean.PlinkUtils.compare_bim* module.

`pyGenClean.MarkerMissingness.snp_missingness.`**`main`**(*argString=None*)

The main function of the module.

Parameters **argString** (*list*) – the options.

These are the steps:

1.Prints the options.

2.Runs Plink with the `geno` option (*runPlink()*).

3.Compares the two `bim` files (before and after the Plink `geno` analysis) (*compareBIM()*).

`pyGenClean.MarkerMissingness.snp_missingness.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list*) – the options.

Returns A argparse.Namespace object created by the argparse module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--geno`	float	The missingness threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *checkArgs()*).

pyGenClean.MarkerMissingness.snp_missingness.**runPlink**(*options*)

Runs Plink with the geno option.

Parameters **options** (*argparse.Namespace*) – the options.

pyGenClean.MarkerMissingness.snp_missingness.**safe_main**()

A safe version of the main function (that catches ProgramError).

pyGenClean.Misc package

Submodules

pyGenClean.Misc.compare_gold_standard module

exception pyGenClean.Misc.compare_gold_standard.**ProgramError**(*msg*)

Bases: exceptions.Exception

An `Exception` raised in case of a problem.

Parameters **msg** (*str*) – the message to print to the user before exiting.

pyGenClean.Misc.compare_gold_standard.**checkArgs**(*args*)

Checks the arguments and options.

Parameters **args** (argparse.Namespace) – a `Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

pyGenClean.Misc.compare_gold_standard.**check_fam_for_samples**(*required_samples*, *source*, *gold*)

Check fam files for required_samples.

pyGenClean.Misc.compare_gold_standard.**computeFrequency**(*prefix*, *outPrefix*)

Compute the frequency using Plink.

pyGenClean.Misc.compare_gold_standard.**compute_statistics**(*out_dir*, *gold_prefix*, *source_prefix*, *same_samples*, *use_sge*, *final_out_prefix*)

Compute the statistics.

`pyGenClean.Misc.compare_gold_standard.`**`exclude_SNPs_samples`**(*inPrefix*, *outPrefix*, *exclusionSNP=None*, *keepSample=None*, *transpose=False*)

Exclude some SNPs and keep some samples using Plink.

`pyGenClean.Misc.compare_gold_standard.`**`extractSNPs`**(*prefixes*, *snpToExtractFileNames*, *outPrefixes*, *runSGE*)

Extract a list of SNPs using Plink.

`pyGenClean.Misc.compare_gold_standard.`**`findFlippedSNPs`**(*goldFrqFile1*, *sourceAlleles*, *outPrefix*)

Find flipped SNPs and flip them in the data1.

`pyGenClean.Misc.compare_gold_standard.`**`findOverlappingSNPsWithGoldStandard`**(*prefix*, *gold_prefixe*, *out_prefix*, *use_marker_names=Fal*

Find the overlapping SNPs in 4 different data sets.

`pyGenClean.Misc.compare_gold_standard.`**`flipSNPs`**(*inPrefix*, *outPrefix*, *flipFileName*)

Flip SNPs using Plink.

`pyGenClean.Misc.compare_gold_standard.`**`illumina_to_snp`**(*strand*, *snp*)

Return the TOP strand of the marker.

Function that takes a strand (TOP or BOT) and a SNP (e.g. : [A/C]) and returns a space separated AlleleA[space]AlleleB string.

Parameters

- **`strand`** (*`str`*) – Either "TOP" or "BOT"
- **`snp`** (*`str`*) – [A/C], [A/T], [G/C], [T/C], [A/G], [C/G], [T/A] or [T/G].

Returns The nucleotide for allele A and the nucleotide for allele B (space separated)

Return type str

`pyGenClean.Misc.compare_gold_standard.`**`keepSamples`**(*prefixes*, *samplesToExtractFileNames*, *outPrefixes*, *runSGE*, *transpose=False*)

Extract a list of SNPs using Plink.

`pyGenClean.Misc.compare_gold_standard.`**`main`**(*argString=None*)

`pyGenClean.Misc.compare_gold_standard.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Returns A `numpy.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (will find the plink binary files by appending the prefix to the `.bim`, `.bed` and `.fam` files, respectively).
`--gold-bfile`	string	The input file prefix (will find the plink binary files by appending the prefix to the `.bim`, `.bed` and `.fam` files, respectively) for the Gold Standard.
`--same-samples`	string	A file containing samples which are present in both the gold standard and the source panel. One line by identity and tab separated. For each row, first sample is Gold Standard, second is source panel.
`--source-manifest`	string	The illumina marker manifest. This file should have tabs as field separator. There should be no lines before the main header line. There should be no lines after the last data line.
`--source-alleles`	string	A file containing the source alleles (TOP). Two columns (separated by tabulation, one with the marker name, the other with the alleles (separated by space). No header.
`--sge`	boolean	Use SGE for parallelization.
`--do-not-flip`	boolean	Do not flip SNPs. WARNING: only use this option only if the Gold Standard was generated using the same chip (hence, flipping is unnecessary).
`--use-marker-names`	boolean	Use marker names instead of (chr, position). WARNING: only use this options only if the Gold Standard was generated using the same chip (hence, they have the same marker names).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.Misc.compare_gold_standard.`**`read_same_samples_file`**`(`*filename*, *out_prefix*`)`

Reads a file containing same samples.

`pyGenClean.Misc.compare_gold_standard.`**`read_source_alleles`**`(`*file_name*`)`

Reads an allele file.

`pyGenClean.Misc.compare_gold_standard.`**`read_source_manifest`**`(`*file_name*`)`

Reads Illumina manifest.

`pyGenClean.Misc.compare_gold_standard.`**`renameSNPs`**`(`*inPrefix*, *updateFileName*, *outPrefix*`)`

Updates the name of the SNPs using Plink.

`pyGenClean.Misc.compare_gold_standard.`**`runCommand`**`(`*command*`)`

Run a command.

`pyGenClean.Misc.compare_gold_standard.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

Module contents

pyGenClean.NoCallHetero package

For more information about how to use this module, refer to the *Clean No Call and Only Heterozygous Markers Module*.

Module contents

Submodules

pyGenClean.NoCallHetero.clean_noCall_hetero_snps module

exception `pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`main`**(*argString=None*)

The main function of the module.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1. Prints the options.
2. Reads the `tfam` and `tped` files and find all heterozygous and all failed markers (*`processTPEDandTFAM()`*).

`pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The input file prefix (Plink tfile).
`--out`	string	The prefix of the output files

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`processTPEDandTFAM`**(*tped, tfam, prefix*)

Process the TPED and TFAM files.

Parameters

- **`tped`** (*`str`*) – the name of the `tped` file.
- **`tfam`** (*`str`*) – the name of the `tfam` file.
- **`prefix`** (*`str`*) – the prefix of the output files.

Copies the original `tfam` file into `prefix.tfam`. Then, it reads the `tped` file and keeps in memory two sets containing the markers which are all failed or which contains only heterozygous genotypes.

It creates two output files, `prefix.allFailed` and `prefix.allHetero`, containing the markers that are all failed and are all heterozygous, respectively.

Note: All heterozygous markers located on the mitochondrial chromosome are not remove.

`pyGenClean.NoCallHetero.clean_noCall_hetero_snps.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.NoCallHetero.heterozygosity_plot module

exception `pyGenClean.NoCallHetero.heterozygosity_plot.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.NoCallHetero.heterozygosity_plot.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.NoCallHetero.heterozygosity_plot.`**`compute_heterozygosity`**`(`*in_prefix*, *nb_samples*`)`

Computes the heterozygosity ratio of samples (from tped).

`pyGenClean.NoCallHetero.heterozygosity_plot.`**`compute_nb_samples`**`(`*in_prefix*`)`

Check the number of samples.

Parameters **`in_prefix`** (*`str`*) – the prefix of the input file.

Returns the number of sample in `prefix.fam`.

`pyGenClean.NoCallHetero.heterozygosity_plot.`**`is_heterozygous`**`(`*genotype*`)`

Tells if a genotype “A B” is heterozygous.

Parameters **`genotype`** (*`str`*) – the genotype to test for heterozygosity.

Returns `True` if the genotype is heterozygous, `False` otherwise.

The genotype must contain two alleles, separated by a space. It then compares the first allele (`genotype[0]`) with the last one (`genotype[-1]`).

```
>>> is_heterozygous("A A")
False
>>> is_heterozygous("G C")
True
>>> is_heterozygous("0 0") # No call is not heterozygous.
False
```

`pyGenClean.NoCallHetero.heterozygosity_plot.`**`main`**`(`*argString=None*`)`

The main function of the module.

Parameters **argString** (*list*) – the options.

These are the steps:

1.Prints the options.

2.Checks the number of samples in the `tfam` file (*compute_nb_samples()*).

3.Computes the heterozygosity rate (*compute_heterozygosity()*).

4.Saves the heterozygosity data (in `out.het`).

5.Plots the heterozygosity rate (*plot_heterozygosity()*).

`pyGenClean.NoCallHetero.heterozygosity_plot.`**parseArgs**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The prefix of the transposed file.
`--boxplot`	bool	Draw a boxplot instead of a histogram.
`--format`	string	The output file format.
`--bins`	int	The number of bins for the histogram.
`--xlim`	float	The limit of the x axis.
`--ymax`	float	“The maximal Y value.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *checkArgs()*).

`pyGenClean.NoCallHetero.heterozygosity_plot.`**plot_heterozygosity**(*heterozygosity*, *options*)

Plots the heterozygosity rate distribution.

Parameters

- **heterozygosity** (*numpy.array*) – the heterozygosity data.
- **options** (*argparse.Namespace*) – the options.

Plots a histogram or a boxplot of the heterozygosity distribution.

`pyGenClean.NoCallHetero.heterozygosity_plot.`**safe_main**()

A safe version of the main function (that catches ProgramError).

`pyGenClean.NoCallHetero.heterozygosity_plot.`**save_heterozygosity**(*heterozygosity*, *samples*, *out_prefix*)

Saves the heterozygosity data.

Parameters

- **heterozygosity** (*numpy.array*) – the heterozygosity data.
- **samples** (*list of tuples of str*) – the list of samples.
- **out_prefix** (*str*) – the prefix of the output files.

pyGenClean.PlateBias package

For more information about how to use this module, refer to the *Plate Bias Module*.

Module contents

Submodules

pyGenClean.PlateBias.plate_bias module

exception `pyGenClean.PlateBias.plate_bias.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.PlateBias.plate_bias.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.PlateBias.plate_bias.`**`computeFrequencyOfSignificantSNPs`**(*options*)

Computes the frequency of the significant markers.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

Extract a list of markers (significant after plate bias analysis) and computes their frequencies.

`pyGenClean.PlateBias.plate_bias.`**`createSummaryFile`**(*results*, *maf*, *prefix*)

Creat the final summary file containing plate bias results.

Parameters

- **`results`** (*`list`*) – the list of all the significant results.
- **`maf`** (*`dict`*) – the minor allele frequency of the significant results.
- **`prefix`** (*`str`*) – the prefix of all the files.

`pyGenClean.PlateBias.plate_bias.`**`executePlateBiasAnalysis`**(*options*)

Execute the plate bias analysis with Plink.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

`pyGenClean.PlateBias.plate_bias.`**`extractSignificantSNPs`**(*prefix*)

Extract significant SNPs in the fisher file.

Parameters **`prefix`** (*`str`*) – the prefix of the input file.

Reads a list of significant markers (`prefix.assoc.fisher`) after plate bias analysis with Plink. Writes a file (`prefix.significant_SNPs.txt`) containing those significant markers.

`pyGenClean.PlateBias.plate_bias.`**`main`**(*argString=None*)

The main function of this module.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1. Runs a plate bias analysis using Plink (*executePlateBiasAnalysis()*).
2. Extracts the list of significant markers after plate bias analysis (*extractSignificantSNPs()*).
3. Computes the frequency of all significant markers after plate bias analysis (*computeFrequencyOfSignificantSNPs()*).

pyGenClean.PlateBias.plate_bias.**parseArgs**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list*) – the options.

Returns A argparse.Namespace object created by the argparse module. It contains the values of the different options.

Options	Type	Description
--bfile	string	The input file prefix (Plink binary).
--loop-assoc	string	The file containing the plate organization of each samples.
--pfilter	float	The significance threshold used for the plate effect.
--out	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *checkArgs()*).

pyGenClean.PlateBias.plate_bias.**runCommand**(*command*)

Run a command.

Parameters **command** (*list*) – the command to run.

Tries to run a command. If it fails, raise a *ProgramError*. This function uses the subprocess module.

Warning: The variable command should be a list of strings (no other type).

pyGenClean.PlateBias.plate_bias.**safe_main**()

A safe version of the main function (that catches ProgramError).

pyGenClean.PlinkUtils package

For more information about how to use this module, refer to the *Plink Utils*.

Module contents

pyGenClean.PlinkUtils.**createRowFromPlinkSpacedOutput**(*line*)

Remove leading spaces and change spaces to tabs.

Param line: a line from a Plink's report file.

Type line: str

Returns an array containing each field from the input line.

Plink's output files are usually created so that they are human readable. Hence, instead of separating fields using tabulation, it uses a certain amount of spaces to create columns. Using the re module, the fields are split.

```
>>> line = " CHR               SNP         BP    A1      A2"
>>> createRowFromPlinkSpacedOutput(line)
['CHR', 'SNP', 'BP', 'A1', 'A2']
```

pyGenClean.PlinkUtils.**get_plink_version**()

Gets the Plink version from the binary.

Returns the version of the Plink software

Return type str

This function uses `subprocess.Popen` to gather the version of the Plink binary. Since executing the software to gather the version creates an output file, it is deleted.

> **Warning:** This function only works as long as the version is returned as `| PLINK! | NNN |` (where, `NNN` is the version), since we use regular expresion to extract the version number from the standard output of the software.

Submodules

pyGenClean.PlinkUtils.compare_bim module

exception pyGenClean.PlinkUtils.compare_bim.**ProgramError**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **msg** (*str*) – the message to print to the user before exiting.

pyGenClean.PlinkUtils.compare_bim.**checkArgs**(*args*)

Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

pyGenClean.PlinkUtils.compare_bim.**compareSNPs**(*before*, *after*, *outFileName*)

Compares two set of SNPs.

Parameters

- **before** (*set*) – the names of the markers in the `before` file.
- **after** (*set*) – the names of the markers in the `after` file.
- **outFileName** (*str*) – the name of the output file.

Finds the difference between two sets of markers, and write them in the `outFileName` file.

> **Note:** A *ProgramError* is raised if:
>
> 1.There are more markers in the `after` set than in the `before` set.
>
> 2.Some markers that are in the `after` set are not in the `before` set.

`pyGenClean.PlinkUtils.compare_bim.`**`main`**`()`

The main function of the module.

The purpose of this module is to find markers that were removed by Plink. When Plinks exclude some markers from binary files, there are no easy way to find the list of removed markers, except by comparing the two BIM files (before and after modification).

Here are the steps of this module:

1. Reads the BIM file before the modification (*`readBIM()`*).
2. Reads the BIM file after the modification (*`readBIM()`*).
3. Compares the list of markers before and after modification, and write the removed markers into a file (*`compareSNPs()`*).

Note: This module only finds marker that were removed (since adding markers to a BIM file usually includes a companion file to tell Plink which marker to add.

`pyGenClean.PlinkUtils.compare_bim.`**`parseArgs`**`()`

Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--before`	string	The name of the BIM file before modification.
`--after`	string	The name of the BIM file after modification.
`--out`	string	The prefix of the output files

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.PlinkUtils.compare_bim.`**`readBIM`**`(`*fileName*`)`

Reads a BIM file.

Parameters **`fileName`** (*`str`*) – the name of the BIM file to read.

Returns the set of markers in the BIM file.

Reads a Plink BIM file and extract the name of the markers. There is one marker per line, and the name of the marker is in the second column. There is no header in the BIM file.

`pyGenClean.PlinkUtils.compare_bim.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.PlinkUtils.plot_MDS module

exception `pyGenClean.PlinkUtils.plot_MDS.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.PlinkUtils.plot_MDS.`**`addCustomOptions`**`(`*parser*`)`

Adds custom options to a parser.

Parameters **`parser`** (*`argparse.parser`*) – the parser.

`pyGenClean.PlinkUtils.plot_MDS.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.PlinkUtils.plot_MDS.`**`extractData`**`(`*fileName*, *populations*`)`

Extract the C1 and C2 columns for plotting.

Parameters

- **`fileName`** (`dict`) – the name of the MDS file.
- **`populations`** – the population of each sample in the MDS file.

Returns the MDS data with information about the population of each sample. The first element of the returned tuple is a tuple. The last element of the returned tuple is the list of the populations (the order is the same as in the first element). The first element of the first tuple is the C1 data, and the last element is the C2 data.

Note: If a sample in the MDS file is not in the population file, it is skip.

`pyGenClean.PlinkUtils.plot_MDS.`**`main`**`()`

The main function of the module.

These are the steps:

1.Reads the population file (`readPopulations()`).

2.Extract the MDS data (`extractData()`).

3.Plots the MDS data (`plotMDS()`).

`pyGenClean.PlinkUtils.plot_MDS.`**`parseArgs`**`()`

Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--file`	string	The MBS file.
`--population-file`	string	A file containing population information.
`--format`	string	The output file format.
`--title`	string	The title of the MDS plot.
`--xlabel`	string	The label of the X axis.
`--ylabel`	string	The label of the Y axis.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.PlinkUtils.plot_MDS.`**`plotMDS`**`(`*data*, *theOrders*, *theLabels*, *theColors*, *theSizes*, *theMarkers*, *options*`)`

Plot the MDS data.

Parameters

- **data** (*list of numpy.array*) – the data to plot (MDS values).
- **theOrders** (*list*) – the order of the populations to plot.
- **theLabels** (*list*) – the names of populations to plot.
- **theColors** (*list*) – the colors of the populations to plot.
- **theSizes** (*list*) – the sizes of the markers for each population to plot.
- **theMarkers** (*list*) – the type of markers for each population to plot.
- **options** (*argparse.Namespace*) – the options.

`pyGenClean.PlinkUtils.plot_MDS.`**`readPopulations`**`(`*inputFileName*`)`

Reads a population file.

Parameters **inputFileName** (*str*) – the name of the population file.

Returns a `dict` of population for each of the samples.

`pyGenClean.PlinkUtils.plot_MDS.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.PlinkUtils.plot_MDS_standalone module

exception `pyGenClean.PlinkUtils.plot_MDS_standalone.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **msg** (*str*) – the message to print to the user before exiting.

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`extractData`**`(`*fileName*, *populations*, *population_order*, *xaxis*, *yaxis*`)`

Extract the C1 and C2 columns for plotting.

Parameters

- **fileName** (*str*) – the name of the MDS file.
- **populations** (*dict*) – the population of each sample in the MDS file.
- **population_order** (*list*) – the required population order.
- **xaxis** (*str*) – the component to print as the X axis.
- **yaxis** (*str*) – the component to print as the Y axis.

Returns the MDS data with information about the population of each sample. The first element of the returned tuple is a tuple. The last element of the returned tuple is the list of the populations (the order is the same as in the first element). The first element of the first tuple is the C1 data, and the last element is the C2 data.

Note: If a sample in the MDS file is not in the population file, it is skip.

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`main`**`()`

The main function of the module.

These are the steps:

1. Reads the population file (`readPopulations()`).
2. Extracts the MDS values (`extractData()`).
3. Plots the MDS values (`plotMDS()`).

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`parseArgs`**`()`

Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--file`	string	The MBS file.
`--population-file`	string	A file containing population information.
`--population-order`	string	The order to print the different populations.
`--population-colors`	string	The population point color in the plot.
`--population-sizes`	string	The population point size in the plot.
`--population-markers`	string	The population point marker in the plot.
`--population-alpha`	string	The population alpha value in the plot.
`--format`	string	The output file format.
`--title`	string	The title of the MDS plot.
`--xaxis`	string	The component to print on the X axis.
`--xlabel`	string	The label of the X axis.
`--yaxis`	string	The component to print on the Y axis.
`--ylabel`	string	The label of the Y axis.
`--legend-position`	string	The position of the legend.
`--legend-size`	int	The size of the legend text.
`--legend-ncol`	int	The number of columns for the legend.
`--legend-alpha`	float	The alpha value of the legend.
`--title-fontsize`	int	The font size of the title.
`--label-fontsize`	int	The font size of the X and Y labels.
`--axis-fontsize`	int	The font size of the X and Y axis.
`--adjust-left`	float	Adjust the left margin.
`--adjust-right`	float	Adjust the right margin.
`--adjust-top`	float	Adjust the top margin.
`--adjust-bottom`	float	Adjust the bottom margin.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`plotMDS`**`(`*data*, *theOrders*, *theLabels*, *theColors*, *theAlphas*, *theSizes*, *theMarkers*, *options*`)`

Plot the MDS data.

Parameters

- **data** (*list of numpy.array*) – the data to plot (MDS values).
- **theOrders** (*list*) – the order of the populations to plot.
- **theLabels** (*list*) – the names of the populations to plot.
- **theColors** (*list*) – the colors of the populations to plot.
- **theAlphas** (*list*) – the alpha value for the populations to plot.
- **theSizes** (*list*) – the sizes of the markers for each population to plot.
- **theMarkers** (*list*) – the type of marker for each population to plot.
- **options** (*argparse.Namespace*) – the options.

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`readPopulations`**(*inputFileName*, *requiredPopulation*)

Reads a population file.

Parameters

- **inputFileName** (*str*) – the name of the population file.
- **requiredPopulation** (*list*) – the required population.

Returns a `dict` containing the population of each samples.

`pyGenClean.PlinkUtils.plot_MDS_standalone.`**`safe_main`**()

A safe version of the main function (that catches ProgramError).

pyGenClean.PlinkUtils.subset_data module

exception `pyGenClean.PlinkUtils.subset_data.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **msg** (*str*) – the message to print to the user before exiting.

`pyGenClean.PlinkUtils.subset_data.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

Note: Only one operation for markers and one operation for samples can be done at a time. Hence, one of `--exclude` or `--extract` can be done for markers, and one of `--remove` or `--keep` can be done for samples.

`pyGenClean.PlinkUtils.subset_data.`**`main`**(*argString=None*)

The main function of the modile.

Parameters **argString** (*list*) – the options.

Here are the steps:

1. Prints the options.
2. Subset the data (*subset_data()*).

Note: The type of the output files are determined by the type of the input files (*e.g.* if the input files are binary files, so will be the output ones).

`pyGenClean.PlinkUtils.subset_data.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the parameters.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ifile`	string	The input file prefix.
`--is-bfile`	bool	The input file is a bfile
`--is-tfile`	bool	The input file is a tfile
`--is-file`	bool	The input file is a file
`--exclude`	string	A file containing SNPs to exclude from the data set.
`--extract`	string	A file containing SNPs to extract from the data set.
`--remove`	string	A file containing samples (FID and IID) to remove from the data set.
`--keep`	string	A file containing samples (FID and IID) to keep from the data set.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.PlinkUtils.subset_data.`**`runCommand`**`(`*command*`)`

Runs a command.

Parameters **`command`** (*`list`*) – the command to run.

If there is a problem, a *`ProgramError`* is raised.

`pyGenClean.PlinkUtils.subset_data.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

`pyGenClean.PlinkUtils.subset_data.`**`subset_data`**`(`*options*`)`

Subset the data.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

Subset the data using either `--exclude` or `--extract`` for markers or ``--remove` or `keep` for samples.

pyGenClean.RelatedSamples package

For more information about how to use this module, refer to the *Related Samples Module*.

Module contents

Submodules

pyGenClean.RelatedSamples.find_related_samples module

exception `pyGenClean.RelatedSamples.find_related_samples.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.RelatedSamples.find_related_samples.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.RelatedSamples.find_related_samples.`**`checkNumberOfSNP`**(*fileName*, *minimumNumber*)

Check there is enough SNPs in the file (with minimum).

Parameters

- **`fileName`** (*`str`*) – the name of the file.
- **`minimumNumber`** (*`int`*) – the minimum number of markers that needs to be in the file.

Returns `True` if there is enough markers in the file, `False` otherwise.

Reads the number of markers (number of lines) in a file.

`pyGenClean.RelatedSamples.find_related_samples.`**`extractRelatedIndividuals`**(*fileName*, *outPrefix*, *ibs2_ratio_threshold*)

Extract related individuals according IBS2* ratio.

Parameters

- **`fileName`** (*`str`*) – the name of the input file.
- **`outPrefix`** (*`str`*) – the prefix of the output files.
- **`ibs2_ratio_threshold`** (*`float`*) – the ibs2 ratio threshold (tells if sample pair is related or not).

Returns a `numpy.recarray` data set containing (for each related sample pair) the `ibs2 ratio`, `Z1`, `Z2` and the type of relatedness.

Reads a `genome` file (provided by *`runGenome()`*) and extract related sample pairs according to `IBS2 ratio`.

A `genome` file contains at least the following information for each sample pair:

- **FID1:** the family ID of the first sample in the pair.
- **IID1:** the individual ID of the first sample in the pair.
- **FID2:** the family ID of the second sample in the pair.
- **IID2:** the individual ID of the second sample in the pair.
- **Z0:** the probability that $IBD = 0$.

- **Z1:** the probability that $IBD = 1$.
- **Z2:** the probability that $IBD = 2$.
- **HOMHOM:** the number of $IBS = 0$ SNP pairs used in `PPC` test.
- **HETHET:** the number of $IBS = 2$ het/het SNP pairs in `PPC` test.

The `IBS2 ratio` is computed using the following formula:

$$\text{IBS2 ratio} = \frac{\text{HETHET}}{\text{HOMHOM} + \text{HETHET}}$$

If the `IBS2 ratio` is higher than the threshold, the samples in the pair are related. The following values help in finding the relatedness of the sample pair.

Values	Relation	Code
$0.17 \leq z_0 \leq 0.33$ and $0.40 \leq z_1 \leq 0.60$	Full-sibs	1
$0.40 \leq z_0 \leq 0.60$ and $0.40 \leq z_1 \leq 0.60$	Half-sibs or Grand-parent-Child or Uncle-Nephew	2
$z_0 \leq 0.05$ and $z_1 \geq 0.95$ and $z_2 \leq 0.05$	Parent-Child	3
$z_0 \leq 0.05$ and $z_1 \leq 0.05$ and $z_2 \geq 0.95$	Twins or Duplicated samples	4

`pyGenClean.RelatedSamples.find_related_samples.`**`extractSNPs`**(*snpsToExtract*, *options*)

Extract markers using Plink.

Parameters

- **`snpsToExtract`** (*`str`*) – the name of the file containing markers to extract.
- **`options`** (*`argparse.Namespace`*) – the options

Returns the prefix of the output files.

`pyGenClean.RelatedSamples.find_related_samples.`**`main`**(*argString=None*)

The main function of this module.

Parameters **`argString`** (*`list`*) – the options.

Here are the steps for this function:

1. Prints the options.
2. Uses Plink to extract markers according to LD (*`selectSNPsAccordingToLD()`*).
3. Checks if there is enough markers after pruning (*`checkNumberOfSNP()`*). If not, then quits.
4. Extract markers according to LD (*`extractSNPs()`*).
5. Runs Plink with the `genome` option (*`runGenome()`*). Quits here if the user asker only for the `genome` file.
6. Finds related individuals and gets values for plotting (*`extractRelatedIndividuals()`*).
7. Plots `Z1` in function of `IBS2 ratio` for related individuals (*`plot_related_data()`*).
8. Plots `Z2` in function of `IBS2 ratio` for related individuals (*`plot_related_data()`*).

`pyGenClean.RelatedSamples.find_related_samples.`**`mergeGenomeLogFiles`**(*outPrefix*, *nbSet*)

Merge genome and log files together.

Parameters

- **`outPrefix`** (*`str`*) – the prefix of the output files.
- **`nbSet`** (*`int`*) – The number of set of files to merge together.

Returns the name of the output file (the `genome` file).

After merging, the files are deleted to save space.

`pyGenClean.RelatedSamples.find_related_samples.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--genome-only`	bool	Only create the genome file.
`--min-nb-snp`	int	The minimum number of markers needed to compute IBS values.
`--indep-pairwise`	string	Three numbers: window size, window shift and the r2 threshold.
`--maf`	string	Restrict to SNPs with MAF >= threshold.
`--ibs2-ratio`	float	The initial IBS2* ratio (the minimum value to show in the plot.
`--sge`	bool	Use SGE for parallelization.
`--sge-walltime`	int	The time limit (for clusters).
`--sge-nodes`	int int	Two INTs (number of nodes and number of processor per nodes).
`--line-per-file-for-sge`	int	The number of line per file for SGE task array.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.RelatedSamples.find_related_samples.`**`plot_related_data`**(*x*, *y*, *code*, *ylabel*, *fileName*, *options*)

Plot Z1 and Z2 in function of IBS2* ratio.

Parameters

- **`x`** (*`numpy.array of floats`*) – the x axis of the plot (`IBS2 ratio`).
- **`y`** (*`numpy.array of floats`*) – the y axis of the plot (either `z1` or `z2`).
- **`code`** (*`numpy.array`*) – the code of the relatedness of each sample pair.
- **`ylabel`** (*`str`*) – the label of the y axis (either `z1` or `z2`).
- **`fileName`** (*`str`*) – the name of the output file.
- **`options`** (*`argparse.Namespace`*) – the options.

There are four different relation codes (represented by 4 different color in the plots:

Code	Relation	Color
1	Full-sbis	`#CC0000`
2	Half-sibs or Grand-parent-Child or Uncle-Nephew	`#0099CC`
3	Parent-Child	`#FF8800`
4	Twins or Duplicated samples	`#9933CC`

Sample pairs with unknown relation are plotted using `#669900` as color.

`pyGenClean.RelatedSamples.find_related_samples.`**`runCommand`**`(`*command*`)`

Run a command.

Parameters **`command`** (*`list`*) – the command to run.

Tries to run a command. If it fails, raise a *`ProgramError`*. This function uses the `subprocess` module.

> **Warning:** The variable `command` should be a list of strings (no other type).

`pyGenClean.RelatedSamples.find_related_samples.`**`runGenome`**`(`*bfile*, *options*`)`

Runs the genome command from plink.

Parameters

- **`bfile`** (*`str`*) – the input file prefix.
- **`options`** (*`argparse.Namespace`*) – the options.

Returns the name of the `genome` file.

Runs Plink with the `genome` option. If the user asks for SGE (`options.sge` is True), a frequency file is first created by plink. Then, the input files are split in `options.line_per_file_for_sge` and Plink is called (using the `genome` option) on the cluster using SGE (*`runGenomeSGE()`*). After the analysis, Plink's output files and logs are merged using *`mergeGenomeLogFiles()`*.

`pyGenClean.RelatedSamples.find_related_samples.`**`runGenomeSGE`**`(`*bfile*, *freqFile*, *nbJob*, *outPrefix*, *options*`)`

Runs the genome command from plink, on SGE.

Parameters

- **`bfile`** (*`str`*) – the prefix of the input file.
- **`freqFile`** (*`str`*) – the name of the frequency file (from Plink).
- **`nbJob`** (*`int`*) – the number of jobs to launch.
- **`outPrefix`** (*`str`*) – the prefix of all the output files.
- **`options`** (*`argparse.Namespace`*) – the options.

Runs Plink with the `genome` options on the cluster (using SGE).

`pyGenClean.RelatedSamples.find_related_samples.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

`pyGenClean.RelatedSamples.find_related_samples.`**`selectSNPsAccordingToLD`**`(`*options*`)`

Compute LD using Plink.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

Returns the name of the output file (from Plink).

`pyGenClean.RelatedSamples.find_related_samples.`**`splitFile`**`(`*inputFileName*, *linePerFile*, *outPrefix*`)`

Split a file.

Parameters

- **`inputFileName`** (*`str`*) – the name of the input file.
- **`linePerFile`** (*`int`*) – the number of line per file (after splitting).
- **`outPrefix`** (*`str`*) – the prefix of the output files.

Returns the number of created temporary files.

Splits a file (`inputFileName` into multiple files containing at most `linePerFile` lines.

pyGenClean.RelatedSamples.merge_related_samples module

exception `pyGenClean.RelatedSamples.merge_related_samples.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.RelatedSamples.merge_related_samples.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – a an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.RelatedSamples.merge_related_samples.`**`main`**(*argString=None*)

The main function of the module.

Parameters **`argString`** (*`list`*) – the options.

`pyGenClean.RelatedSamples.merge_related_samples.`**`merge_related_samples`**(*file_name*, *out_prefix*, *no_status*)

Merge related samples.

Parameters

- **`file_name`** (*`str`*) – the name of the input file.
- **`out_prefix`** (*`str`*) – the prefix of the output files.
- **`no_status`** (*`boolean`*) – is there a status column in the file?

In the output file, there are a pair of samples per line. Hence, one can find related individuals by merging overlapping pairs.

`pyGenClean.RelatedSamples.merge_related_samples.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ibs-related`	string	The input file containing related individuals according to IBS value.
`--no-status`	bool	The input file doesn't have a `status` column.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.RelatedSamples.merge_related_samples.`**`safe_main`**()

A safe version of the main function (that catches ProgramError).

pyGenClean.SampleMissingness package

For more information about how to use this module, refer to the *Sample Missingness Module*.

Module contents

Submodules

pyGenClean.SampleMissingness.sample_missingness module

exception `pyGenClean.SampleMissingness.sample_missingness.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.SampleMissingness.sample_missingness.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – a object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.SampleMissingness.sample_missingness.`**`main`**`(`*argString=None*`)`

The main function of the module.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1.Prints the options.

2.Runs Plink with the `mind` option (*`runPlink()`*).

`pyGenClean.SampleMissingness.sample_missingness.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ifile`	string	The input file prefix (either a Plink binary file or a tfile).
`--is-bfile`	bool	The input file (`--ifile`) is a bfile instead of a tfile.
`--mind`	float	The missingness threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.SampleMissingness.sample_missingness.`**`runPlink`**`(`*options*`)`

Run Plink with the `mind` option.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

`pyGenClean.SampleMissingness.sample_missingness.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.SexCheck package

For more information about how to use this module, refer to the *Sex Check Module*.

Module contents

Submodules

pyGenClean.SexCheck.baf_lrr_plot module

exception `pyGenClean.SexCheck.baf_lrr_plot.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (`str`) – the message to print to the user before exiting.

`pyGenClean.SexCheck.baf_lrr_plot.`**`checkArgs`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.SexCheck.baf_lrr_plot.`**`check_file_names`**`(`*samples*, *raw_dir*, *options*`)`

Check if all files are present.

Parameters

- **`samples`** (`list of tuples`) – a list of tuples with the family ID as first element (str) and sample ID as last element (str).
- **`raw_dir`** (`str`) – the directory containing the raw files.
- **`options`** (`argparse.Namespace`) – the options.

Returns a dict containing samples as key (a tuple with the family ID as first element and sample ID as last element) and the name of the raw file as element.

`pyGenClean.SexCheck.baf_lrr_plot.`**`encode_chromosome`**`(`*chromosome*`)`

Encodes chromosomes.

Parameters **`chromosome`** (`str`) – the chromosome to encode.

Returns the encoded chromosome.

Encodes the sexual chromosomes, from `23` and `24` to `X` and `Y`, respectively.

Note: Only the sexual chromosomes are encoded.

```
>>> encode_chromosome("23")
'X'
>>> encode_chromosome("24")
'Y'
>>> encode_chromosome("This is not a chromosome")
'This is not a chromosome'
```

`pyGenClean.SexCheck.baf_lrr_plot.`**`main`**`(`*argString=None*`)`

The main function of this module.

Parameters **`argString`** (*`list`*) – the options.

These are the steps:

1.Prints the options.

2.Reads the problematic samples (*`read_problematic_samples()`*).

3.Finds and checks the raw files for each of the problematic samples (*`check_file_names()`*).

4.Plots the BAF and LRR (*`plot_baf_lrr()`*).

`pyGenClean.SexCheck.baf_lrr_plot.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--problematic-samples`	string	The list of sample with sex problems to plot
`--use-full-ids`	bool	Use full sample IDs (famID and indID).
`--full-ids-delimiter`	string	The delimiter between famID and indID.
`--raw-dir`	string	Directory containing information about every samples (BAF and LRR).
`--format`	string	The output file format (png, ps, pdf, or X11).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see *`checkArgs()`*).

`pyGenClean.SexCheck.baf_lrr_plot.`**`plot_baf_lrr`**`(`*file_names*, *options*`)`

Plot BAF and LRR for a list of files.

Parameters

- **`file_names`** (*`dict`*) – contains the name of the input file for each sample.
- **`options`** (*`argparse.Namespace`*) – the options.

Plots the BAF (B Allele Frequency) and LRR (Log R Ratio) of each samples. Only the sexual chromosome are shown.

`pyGenClean.SexCheck.baf_lrr_plot.`**`read_problematic_samples`**`(`*file_name*`)`

Reads a file with sample IDs.

Parameters **`file_name`** (*`str`*) – the name of the file containing problematic samples after sex check.

Returns a set of problematic samples (tuple containing the family ID as first element and the sample ID as last element).

Reads a file containing problematic samples after sex check. The file is provided by the module *pyGenClean.SexCheck.sex_check*. This file contains two columns, the first one being the family ID and the second one, the sample ID.

`pyGenClean.SexCheck.baf_lrr_plot.`**`safe_main`**`()`
A safe version of the main function (that catches ProgramError).

pyGenClean.SexCheck.gender_plot module

exception `pyGenClean.SexCheck.gender_plot.`**`ProgramError`**`(`*msg*`)`
Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*str*) – the message to print to the user before exiting.

`pyGenClean.SexCheck.gender_plot.`**`checkArgs`**`(`*args*`)`
Checks the arguments and options.

Parameters **`args`** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *ProgramError* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.SexCheck.gender_plot.`**`encode_chr`**`(`*chromosome*`)`
Encodes chromosomes.

Parameters **`chromosome`** (*str*) – the chromosome to encode.

Returns the encoded chromosome as `int`.

It changes `X`, `Y`, `XY` and `MT` to `23`, `24`, `25` and `26`, respectively. It changes everything else as `int`.

If `ValueError` is raised, then *ProgramError* is also raised. If a chromosome as a value below 1 or above 26, a *ProgramError* is raised.

```
>>> [encode_chr(str(i)) for i in range(0, 11)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> [encode_chr(str(i)) for i in range(11, 21)]
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> [encode_chr(str(i)) for i in range(21, 27)]
[21, 22, 23, 24, 25, 26]
>>> [encode_chr(i) for i in ["X", "Y", "XY", "MT"]]
[23, 24, 25, 26]
>>> encode_chr("27")
Traceback (most recent call last):
    ...
ProgramError: 27: invalid chromosome
>>> encode_chr("XX")
Traceback (most recent call last):
    ...
ProgramError: XX: invalid chromosome
```

`pyGenClean.SexCheck.gender_plot.`**`encode_gender`**`(`*gender*`)`
Encodes the gender.

Parameters **`gender`** (*str*) – the gender to encode.

Returns the encoded gender.

It changes `1` and `2` to `Male` and `Female` respectively. It encodes everything else to `Unknown`.

```
>>> encode_gender("1")
'Male'
>>> encode_gender("2")
'Female'
>>> encode_gender("0")
'Unknown'
>>> encode_gender("This is not a gender code")
'Unknown'
```

`pyGenClean.SexCheck.gender_plot.`**`main`**`(`*argString=None*`)`

The main function of the module.

Parameters **`argString`** (`list`) – the options.

These are the steps:

1.Prints the options.

2.If there are `summarized_intensities` provided, reads the files (`read_summarized_intensities()`) and skips to step 7.

3.Reads the `bim` file to get markers on the sexual chromosomes (`read_bim()`).

4.Reads the `fam` file to get gender (`read_fam()`).

5.Reads the file containing samples with sex problems (`read_sex_problems()`).

6.Reads the intensities and summarizes them (`read_intensities()`).

7.Plots the summarized intensities (`plot_gender()`).

`pyGenClean.SexCheck.gender_plot.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (`list`) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The plink binary file containing information about markers and individuals.
`--intensities`	string	A file containing alleles intensities for each of the markers located on the X and Y chromosome.
`--summarized-intensities`	string	The prefix of six files containing summarized chr23 and chr24 intensities.
`--sex-problems`	string	The file containing individuals with sex problems.
`--format`	string	The output file format (png, ps, pdf, or X11).
`--xlabel`	string	The label of the X axis.
`--ylabel`	string	The label of the Y axis.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.SexCheck.gender_plot.`**`plot_gender`**`(`*data*, *options*`)`

Plots the gender.

Parameters

- **data** (*numpy.recarray*) – the data to plot.
- **options** (*argparse.Namespace*) – the options.

Plots the summarized intensities of the markers on the Y chromosomes in function of the markers on the X chromosomes, with problematic samples with different colors.

Also uses `print_data_to_file()` to save the data, so that it is faster to rerun the analysis.

`pyGenClean.SexCheck.gender_plot.`**`print_data_to_file`**`(`*data*, *file_name*`)`

Prints data to file.

Parameters

- **file_name** (*str*) – the name of the output file.

is preceded by

- **data** (*numpy.recarray*) – the data to print.

`pyGenClean.SexCheck.gender_plot.`**`read_bim`**`(`*file_name*`)`

Reads the BIM file to gather marker names.

Parameters **file_name** (*str*) – the name of the `bim` file.

Returns a `dict` containing the chromosomal location of each marker on the sexual chromosomes.

It uses the `encode_chr()` to encode the chromosomes from `X` and `Y` to `23` and `24`, respectively.

`pyGenClean.SexCheck.gender_plot.`**`read_fam`**`(`*file_name*`)`

Reads the FAM file to gather sample names.

Parameters **file_name** (*str*) – the `fam` file to read.

Returns a `dict` containing the gender of each samples.

It uses the `encode_gender()` to encode the gender from `` 1``and``2 `` to `Male` and `Female`, respectively.

`pyGenClean.SexCheck.gender_plot.`**`read_intensities`**`(`*file_name*, *needed_markers_chr*, *needed_samples_gender*, *sex_problems*`)`

Reads the intensities from a file.

Parameters

- **file_name** (*str*) – the name of the input file.
- **needed_markers_chr** (*dict*) – the markers that are needed.
- **needed_samples_gender** (*dict*) – the gender of all the samples.
- **sex_problems** (*frozenset*) – the sample with sex problem.

Returns a :py:class‘numpy.recarray‘ containing the following columns (for each of the samples): `sampleID`, `chr23`, `chr24`, `gender` and `status`.

Reads the normalized intensities from a final report. The file must contain the following columns: `SNP Name`, `Sample ID`, `X`, `Y` and `Chr`. It then keeps only the required markers (those that are on sexual chromosomes (`23` or `24`), encoding *NaN* intensities to zero.

The final data set contains the following information for each sample:

- `sampleID`: the sample ID.
- `chr23`: the summarized intensities for chromosome 23.
- `chr24`: the summarized intensities for chromosome 24.
- `gender`: the gender of the sample (`Male` or `Female`).
- `status`: the status of the sample (`OK` or `Problem`).

The summarized intensities for a chromosome (S_{chr}) is computed using this formula (where I_{chr} is the set of all marker intensities on chromosome *chr*):

$$S_{chr} = \frac{\sum I_{chr}}{||I_{chr}||}$$

`pyGenClean.SexCheck.gender_plot.`**`read_sex_problems`**`(`*file_name*`)`

Reads the sex problem file.

Parameters **`file_name`** (`str`) – the name of the file containing sex problems.

Returns a `frozenset` containing samples with sex problem.

If there is no `file_name` (*i.e.* is `None`), then an empty `frozenset` is returned.

`pyGenClean.SexCheck.gender_plot.`**`read_summarized_intensities`**`(`*prefix*`)`

Reads the summarized intensities from 6 files.

Parameters **`prefix`** (`str`) – the prefix of the six files.

Returns a :py:class‘numpy.recarray‘ containing the following columns (for each of the samples): `sampleID`, `chr23`, `chr24`, `gender` and `status`.

Instead of reading a final report (like *`read_intensities()`*), this function reads six files previously created by this module to gather sample information. Here are the content of the six files:

- `prefix.ok_females.txt`: information about females without sex problem.
- `prefix.ok_males.txt`: information about males without sex problem.
- **`prefix.ok_unknowns.txt`: information about unknown gender without sex** problem.
- **`prefix.problematic_females.txt`: information about females with sex** problem.
- **`prefix.problematic_males.txt`: information about males with sex** problem.
- **`prefix.problematic_unknowns.txt`: information about unknown gender** with sex problem.

Each file contains the following columns: `sampleID`, `chr23`, `chr24`, `gender` and `status`.

The final data set contains the following information for each sample:

- `sampleID`: the sample ID.
- `chr23`: the summarized intensities for chromosome 23.
- `chr24`: the summarized intensities for chromosome 24.
- `gender`: the gender of the sample (`Male` or `Female`).
- `status`: the status of the sample (`OK` or `Problem`).

The summarized intensities for a chromosome (S_{chr}) is computed using this formula (where I_{chr} is the set of all marker intensities on chromosome *chr*):

$$S_{chr} = \frac{\sum I_{chr}}{||I_{chr}||}$$

`pyGenClean.SexCheck.gender_plot.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

pyGenClean.SexCheck.sex_check module

exception `pyGenClean.SexCheck.sex_check.`**`ProgramError`**(*msg*)

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

`pyGenClean.SexCheck.sex_check.`**`checkArgs`**(*args*)

Checks the arguments and options.

Parameters **`args`** (*`argparse.Namespace`*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the *`ProgramError`* class, a message is printed to the `sys.stderr` and the program exists with code 1.

`pyGenClean.SexCheck.sex_check.`**`checkBim`**(*fileName*, *minNumber*, *chromosome*)

Checks the BIM file for chrN markers.

Parameters

- **`fileName`** (*`str`*) –
- **`minNumber`** (*`int`*) –
- **`chromosome`** (*`str`*) –

Returns `True` if there are at least `minNumber` markers on chromosome `chromosome`, `False` otherwise.

`pyGenClean.SexCheck.sex_check.`**`computeHeteroPercentage`**(*fileName*)

Computes the heterozygosity percentage.

Parameters **`fileName`** (*`str`*) – the name of the input file.

Reads the `ped` file created by Plink using the `recodeA` options (see *`createPedChr23UsingPlink()`*) and computes the heterozygosity percentage on the chromosome `23`.

`pyGenClean.SexCheck.sex_check.`**`computeNoCall`**(*fileName*)

Computes the number of no call.

Parameters **`fileName`** (*`str`*) – the name of the file

Reads the `ped` file created by Plink using the `recodeA` options (see *`createPedChr24UsingPlink()`*) and computes the number and percentage of no calls on the chromosome `24`.

`pyGenClean.SexCheck.sex_check.`**`createGenderPlot`**(*bfile*, *intensities*, *problematic_samples*, *format*, *out_prefix*)

Creates the gender plot.

Parameters

- **`bfile`** (*`str`*) – the prefix of the input binary file.
- **`intensities`** (*`str`*) – the file containing the intensities.
- **`problematic_samples`** (*`str`*) – the file containing the problematic samples.
- **`format`** (*`str`*) – the format of the output plot.
- **`out_prefix`** (*`str`*) – the prefix of the output file.

Creates the gender plot of the samples using the *`pyGenClean.SexCheck.gender_plot`* module.

`pyGenClean.SexCheck.sex_check.`**`createLrrBafPlot`**`(`*raw_dir*, *problematic_samples*, *format*, *dpi*, *out_prefix*`)`

Creates the LRR and BAF plot.

Parameters

- **`raw_dir`** (*`str`*) – the directory containing the intensities.
- **`problematic_samples`** (*`str`*) – the file containing the problematic samples.
- **`format`** (*`str`*) – the format of the plot.
- **`dpi`** – the DPI of the resulting images.
- **`out_prefix`** (*`str`*) – the prefix of the output file.

Creates the LRR (Log R Ratio) and BAF (B Allele Frequency) of the problematic samples using the *`pyGenClean.SexCheck.baf_lrr_plot`* module.

`pyGenClean.SexCheck.sex_check.`**`createPedChr23UsingPlink`**`(`*options*`)`

Run Plink to create a ped format.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

Uses Plink to create a `ped` file of markers on the chromosome `23`. It uses the `recodeA` options to use additive coding. It also subsets the data to keep only samples with sex problems.

`pyGenClean.SexCheck.sex_check.`**`createPedChr24UsingPlink`**`(`*options*`)`

Run plink to create a ped format.

Parameters **`options`** (*`argparse.Namespace`*) – the options.

Uses Plink to create a `ped` file of markers on the chromosome `24`. It uses the `recodeA` options to use additive coding. It also subsets the data to keep only samples with sex problems.

`pyGenClean.SexCheck.sex_check.`**`main`**`(`*argString=None*`)`

The main function of the module.

Parameters **`argString`** (*`list`*) – the options.

These are the following steps:

1. Prints the options.
2. Checks if there are enough markers on the chromosome `23` (*`checkBim()`*). If not, quits here.
3. Runs the sex check analysis using Plink (*`runPlinkSexCheck()`*).
4. If there are no sex problems, then quits (*`readCheckSexFile()`*).
5. Creates the recoded file for the chromosome `23` (*`createPedChr23UsingPlink()`*).
6. Computes the heterozygosity percentage on the chromosome `23` (*`computeHeteroPercentage()`*).
7. If there are enough markers on chromosome `24` (at least 1), creates the recoded file for this chromosome (*`createPedChr24UsingPlink()`*).
8. Computes the number of no call on the chromosome `24` (*`computeNoCall()`*).
9. If required, plots the gender plot (*`createGenderPlot()`*).
10. If required, plots the BAF and LRR plot (*`createLrrBafPlot()`*).

`pyGenClean.SexCheck.sex_check.`**`parseArgs`**`(`*argString=None*`)`

Parses the command line options and arguments.

Parameters **`argString`** (*`list`*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary).
`--femaleF`	float	The female F threshold.
`--maleF`	float	The male F threshold.
`--nbChr23`	int	The minimum number of markers on chromosome 23 before computing Plink's sex check.
`--gender-plot`	bool	Create the gender plot.
`--sex-chr-intensities`	string	A file containing alleles intensities for each of the markers located on the X and Y chromosome.
`--gender-plot-format`	string	The output file format for the gender plot.
`--lrr-baf`	bool	Create the LRR and BAF plot.
`--lrr-baf-raw-dir`	string	Directory containing information about every samples (BAF and LRR).
`--lrr-baf-format`	string	The output file format.
`--lrr-baf-dpi`	int	The pixel density of the figure(s) (DPI).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`pyGenClean.SexCheck.sex_check.`**`readCheckSexFile`**(*fileName*, *allProblemsFileName*, *idsFileName*, *femaleF*, *maleF*)

Reads the Plink check-sex output file.

Parameters

- **`fileName`** (`str`) – the name of the input file.
- **`allProblemsFileName`** (`str`) – the name of the output file that will contain all the problems.
- **`idsFileName`** (`str`) – the name of the output file what will contain samples with sex problems.
- **`femaleF`** (`float`) – the F threshold for females.
- **`maleF`** (`float`) – the F threshold for males.

Returns `True` if there are sex problems, `False` otherwise.

Reads sex check file provided by `runPlinkSexCheck()` (Plink) and extract the samples that have sex problems.

`pyGenClean.SexCheck.sex_check.`**`runCommand`**(*command*)

Run a command.

Parameters **`command`** (`list`) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable command should be a list of strings (no other type).

`pyGenClean.SexCheck.sex_check.`**`runPlinkSexCheck`**(*options*)

Runs Plink to perform a sex check analysis.

Parameters **`options`** (`argparse.Namespace`) – the options.

Uses Plink to perform a sex check analysis.

`pyGenClean.SexCheck.sex_check.`**`safe_main`**`()`

A safe version of the main function (that catches ProgramError).

Submodules

pyGenClean.pipeline_error module

exception `pyGenClean.pipeline_error.`**`ProgramError`**`(`*msg*`)`

Bases: `exceptions.Exception`

An `Exception` raised in case of a problem.

Parameters **`msg`** (*`str`*) – the message to print to the user before exiting.

pyGenClean.run_data_clean_up module

`pyGenClean.run_data_clean_up.`**`all_files_exist`**`(`*file_list*`)`

Check if all files exist.

Parameters **`file_list`** (*`list`*) – the names of files to check.

Returns `True` if all files exist, `False` otherwise.

`pyGenClean.run_data_clean_up.`**`check_args`**`(`*args*`)`

Checks the arguments and options.

Parameters **`args`** (`argparse.Namespace`) – an object containing the options and arguments of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exits with error code 1.

`pyGenClean.run_data_clean_up.`**`check_input_files`**`(`*prefix*, *the_type*, *required_type*`)`

Check that the file is of a certain file type.

Parameters

- **`prefix`** (*`str`*) – the prefix of the input files.
- **`the_type`** (*`str`*) – the type of the input files (bfile, tfile or file).
- **`required_type`** (*`str`*) – the required type of the input files (bfile, tfile or file).

Returns `True` if everything is OK.

Checks if the files are of the required type, according to their current type. The available types are `bfile` (binary), `tfile` (transposed) and `file` (normal).

`pyGenClean.run_data_clean_up.`**`count_markers_samples`**`(`*prefix*, *file_type*`)`

Counts the number of markers and samples in plink file.

Parameters

- **`prefix`** (*`str`*) – the prefix of the files.
- **`file_type`** (*`str`*) – the file type.

Returns the number of markers and samples (in a tuple).

pyGenClean.run_data_clean_up.**main**()

The main function.

These are the steps performed for the data clean up:

1.Prints the version number.

2.Reads the configuration file (*read_config_file()*).

3.Creates a new directory with data_clean_up as prefix and the date and time as suffix.

4.Check the input file type (bfile, tfile or file).

5.Creates an intermediate directory with the section as prefix and the script name as suffix (inside the previous directory).

6.Runs the required script in order (according to the configuration file section).

Note: The main function is not responsible to check if the required files exist. This should be done in the run functions.

pyGenClean.run_data_clean_up.**parse_args**()

Parses the command line options and arguments.

Returns A argparse.Namespace object created by the argparse module. It contains the values of the different options.

Options	Type	Description
--bfile	String	The input binary file prefix from Plink.
--tfile	String	The input transposed file prefix from Plink.
--file	String	The input file prefix from Plink.
--conf	String	The parameter file for the data clean up.
--report-author	String	The current project number.
--report-number	String	The current project author.
--report-background	String	Text of file containing the background section of the report.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see checkArgs()).

pyGenClean.run_data_clean_up.**read_config_file**(*filename*)

Reads the configuration file.

Parameters **filename** (*str*) – the name of the file containing the configuration.

Returns A tuple where the first element is a list of sections, and the second element is a map containing the configuration (options and values).

The structure of the configuration file is important. Here is an example of a configuration file:

```
[1] # Computes statistics on duplicated samples
script = duplicated_samples

[2] # Removes samples according to missingness
script = sample_missingness

[3] # Removes markers according to missingness
script = snp_missingness

[4] # Removes samples according to missingness (98%)
```

```
script = sample_missingness
mind = 0.02

[5] # Performs a sex check
script = sex_check

[6] # Flags markers with MAF=0
script = flag_maf_zero

[7] # Flags markers according to Hardy Weinberg
script = flag_hw

[8] # Subset the dataset (excludes markers and remove samples)
script = subset
exclude = .../filename
rempove = .../filename
```

Sections are in square brackets and must be `integer`. The section number represent the step at which the script will be run (*i.e.* from the smallest number to the biggest). The sections must be continuous.

Each section contains the script names (`script` variable) and options of the script (all other variables) (*e.g.* section 4 runs the `sample_missingness` script (*run_sample_missingness()*) with option `mind` sets to 0.02).

Here is a list of the available scripts:

- `duplicated_samples` (*run_duplicated_samples()*)
- `duplicated_snps` (*run_duplicated_snps()*)
- `noCall_hetero_snps` (*run_noCall_hetero_snps()*)
- `sample_missingness` (*run_sample_missingness()*)
- `snp_missingness` (*run_snp_missingness()*)
- `sex_check` (*run_sex_check()*)
- `plate_bias` (*run_plate_bias()*)
- `contamination` (*run_contamination()*)
- `remove_heterozygous_haploid` (*run_remove_heterozygous_haploid()*)
- `find_related_samples` (*run_find_related_samples()*)
- `check_ethnicity` (*run_check_ethnicity()*)
- `flag_maf_zero` (*run_flag_maf_zero()*)
- `flag_hw` (*run_flag_hw()*)
- `subset` (*run_subset_data()*)
- `compare_gold_standard` (*run_compare_gold_standard()*)

`pyGenClean.run_data_clean_up.`**`run_check_ethnicity`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Runs step10 (check ethnicity).

Parameters

- **`in_prefix`** (*str*) – the prefix of the input files.
- **`in_type`** (*str*) – the type of the input files.

- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.Ethnicity.check_ethnicity* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.Ethnicity.check_ethnicity* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_command`**(*command*)

Run a command using subprocesses.

Parameters **command** (*list*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`.

Warning: The variable `command` should be a list of strings (no other type).

`pyGenClean.run_data_clean_up.`**`run_compare_gold_standard`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Compares with a gold standard data set (compare_gold_standard.

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.Misc.compare_gold_standard* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.Misc.compare_gold_standard* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_contamination`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Runs the contamination check for samples.

Parameters

- **in_prefix** (*str*) – the prefix of the input files.

- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

`pyGenClean.run_data_clean_up.`**`run_duplicated_samples`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Runs step1 (duplicated samples).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the *pyGenClean.DupSamples.duplicated_samples* module. The required file type for this module is `tfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

`pyGenClean.run_data_clean_up.`**`run_duplicated_snps`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Runs step2 (duplicated snps).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the *pyGenClean.DupSNPs.duplicated_snps* module. The required file type for this module is `tfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: This function creates a `map` file, needed for the *pyGenClean.DupSNPs.duplicated_snps* module.

`pyGenClean.run_data_clean_up.`**`run_find_related_samples`**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Runs step9 (find related samples).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.RelatedSamples.find_related_samples* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.RelatedSamples.find_related_samples* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_flag_hw`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step12 (flag HW).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.FlagHW.flag_hw* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.FlagHW.flag_hw* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_flag_maf_zero`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step11 (flag MAF zero).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.FlagMAF.flag_maf_zero* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.FlagMAF.flag_maf_zero* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_noCall_hetero_snps`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step 3 (clean no call and hetero).

Parameters

- **`in_prefix`** (*str*) – the prefix of the input files.
- **`in_type`** (*str*) – the type of the input files.
- **`out_prefix`** (*str*) – the output prefix.
- **`base_dir`** (*str*) – the output directory.
- **`options`** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the *pyGenClean.NoCallHetero.clean_noCall_hetero_snps* module. The required file type for this module is `tfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

`pyGenClean.run_data_clean_up.`**`run_plate_bias`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step7 (plate bias).

Parameters

- **`in_prefix`** (*str*) – the prefix of the input files.
- **`in_type`** (*str*) – the type of the input files.
- **`out_prefix`** (*str*) – the output prefix.
- **`base_dir`** (*str*) – the output directory.
- **`options`** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.PlateBias.plate_bias* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.PlateBias.plate_bias* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`pyGenClean.run_data_clean_up.`**`run_remove_heterozygous_haploid`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step8 (remove heterozygous haploid).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.HeteroHap.remove_heterozygous_haploid* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

`pyGenClean.run_data_clean_up.`**`run_sample_missingness`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step4 (clean mind).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.SampleMissingness.sample_missingness* module. The required file type for this module is either a `bfile` or a `tfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

`pyGenClean.run_data_clean_up.`**`run_sex_check`**`(`*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*`)`

Runs step6 (sexcheck).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the *pyGenClean.SexCheck.sex_check* module. The required file type for this module is `bfile`, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The *pyGenClean.SexCheck.sex_check* module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

pyGenClean.run_data_clean_up.**run_snp_missingness**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Run step5 (clean geno).

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (bfile).

This function calls the *pyGenClean.MarkerMissingness.snp_missingness* module. The required file type for this module is bfile, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

pyGenClean.run_data_clean_up.**run_subset_data**(*in_prefix*, *in_type*, *out_prefix*, *base_dir*, *options*)

Subsets the data.

Parameters

- **in_prefix** (*str*) – the prefix of the input files.
- **in_type** (*str*) – the type of the input files.
- **out_prefix** (*str*) – the output prefix.
- **base_dir** (*str*) – the output directory.
- **options** (*list*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (bfile).

This function calls the pyGenClean.pyGenClean.PlinkUtils.subset_data module. The required file type for this module is bfile, hence the need to use the *check_input_files()* to check if the file input file type is the good one, or to create it if needed.

Note: The output file type is the same as the input file type.

pyGenClean.run_data_clean_up.**safe_main**()

A safe version of the main function (that catches ProgramError).

pyGenClean.version module

Module contents

pyGenClean.**add_file_handler_to_root**(*log_fn*)

Adds a file handler to the root logging.

Parameters **log_fn** (*str*) – the name of the log file.

5.2 Contamination Module

The usage of the standalone module is shown below:

```
$ pyGenClean_check_contamination --help
usage: pyGenClean_check_contamination [-h] [-v] --bfile FILE --raw-dir DIR
                                      [--colsample COL] [--colmarker COL]
                                      [--colbaf COL] [--colab1 COL]
                                      [--colab2 COL] [--sge]
                                      [--sge-walltime TIME]
                                      [--sge-nodes INT INT]
                                      [--sample-per-run-for-sge INT]
                                      [--out FILE]

Check BAF and LogR ratio for data contamination.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively).

Raw Data:
  --raw-dir DIR         Directory containing the raw data (one file per
                        sample, where the name of the file (minus the
                        extension) is the sample identification number.
  --colsample COL       The sample column. [default: Sample Name]
  --colmarker COL       The marker column. [default: SNP Name]
  --colbaf COL          The B allele frequency column. [default: B Allele
                        Freq]
  --colab1 COL          The AB Allele 1 column. [default: Allele1 - AB]
  --colab2 COL          The AB Allele 2 column. [default: Allele2 - AB]

SGE Options:
  --sge                 Use SGE for parallelization.
  --sge-walltime TIME   The walltime for the job to run on the cluster. Do not
                        use if you are not required to specify a walltime for
                        your jobs on your cluster (e.g. 'qsub
                        -lwalltime=1:0:0' on the cluster).
  --sge-nodes INT INT   The number of nodes and the number of processor per
                        nodes to use (e.g. 'qsub -lnodes=X:ppn=Y' on the
                        cluster, where X is the number of nodes and Y is the
                        number of processor to use. Do not use if you are not
                        required to specify the number of nodes for your jobs
                        on the cluster.
  --sample-per-run-for-sge INT
```

```
                        The number of sample to run for a single SGE job.
                        [default: 30]

Output File:
  --out FILE            The prefix of the output files. [default:
                        contamination]
```

5.2.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest). It also uses intensities file (one per sample to test) usually provided by the genotyping platform.

5.2.2 Procedure

Here are the steps performed by the module:

1. Selects only markers located on autosomes.
2. Compute frequency for each autosomal markers (as required by *bafRegress*).
3. Execute *bafRegress* on the dataset (in parallel if required).

5.2.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `contamination`]):

1. `contamination.to_extract`: the autosomal markers that will be used by *bafRegress*.
2. `contamination.frq`: the frequency of each of the autosomal markers.
3. **`contamination.bafRegress`: the *bafRegress* results for each of the** tested sample.

5.2.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *`pyGenClean.Contamination.contamination`*

5.3 Duplicated Samples Module

The usage of the standalone module is shown below:

```
$ pyGenClean_duplicated_samples --help
usage: pyGenClean_duplicated_samples [-h] [-v] --tfile FILE
                                     [--sample-completion-threshold FLOAT]
                                     [--sample-concordance-threshold FLOAT]
                                     [--out FILE]

Extracts and merges duplicated samples.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --tfile FILE          The input file prefix (will find the tped and tfam
                        file by appending the prefix to .tped and .tfam,
                        respectively.) The duplicated samples should have the
                        same identification numbers (both family and
                        individual ids.)

Options:
  --sample-completion-threshold FLOAT
                        The completion threshold to consider a replicate when
                        choosing the best replicates and for creating the
                        composite samples. [default: 0.9]
  --sample-concordance-threshold FLOAT
                        The concordance threshold to consider a replicate when
                        choosing the best replicates and for creating the
                        composite samples. [default: 0.97]

Output File:
  --out FILE            The prefix of the output files. [default: dup_samples]
```

5.3.1 Input Files

This module uses PLINK's transposed pedfile format (`tped` and `tfam` files). For this step to work, the duplicated samples must have the same identification (family and sample ID). One should keep a file containing the original identifications before modifying the dataset accordingly.

5.3.2 Procedure

Here are the steps performed by the module:

1. Reads the `tfam` file to find duplicated samples.
2. Separates the duplicated samples from the unique samples.
3. Writes the unique samples into a file.
4. Reads the `tped` file and write the pedigree file for the unique samples. Saves in memory the pedigree for the duplicated samples. Updates the indexes of the duplicated samples.
5. If there are no duplicated samples, simply create the final file. Stop here.
6. Computes the completion (for each of the duplicated samples) and the concordance of each sample pairs.
7. Prints statistics (concordance and completion).
8. Prints the concordance matrix for each duplicated samples.
9. Prints the `tped` and the `tfam` file for the duplicated samples.
10. Chooses the best of each duplicates (to keep and to complete) according to completion and concordance.
11. Creates a unique `tped` and `tfam` from the duplicated samples by completing the best chosen one with the other samples.
12. Creates the final dataset.

5.3.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* dup_samples]):

1. No output file is created.
2. No output file is created.
3. Only one of the two PLINK's transposed pedfiles is created:
   - `dup_samples.unique_samples.tfam`: the `tfam` file containing only the unique samples from the original dataset.
4. The second of the two PLINK's transposed pedfiles is created (see previous step):
   - `dup_samples.unique_samples.tped`: the `tped` file containing only the unique samples from the original dataset.
5. If there are not duplicated samples, the final PLINK's transposed pedfiles are created (if not, continue tu next step):
   - `dup_samples.final`: the `tfam` and `tped` final files.
6. One result file is created:
   - `dup_samples.diff`: a file containing the differences in the genotypes for each pair of duplicated samples. Each line contains the following information: `name` the name of the marker, `famID` the family ID, `indID` the individual ID, `dupIndex_1` the index of the first duplicated sample in the original dataset (since the identification of each duplicated samples are the same), `dupIndex_2` the index of the second duplicated sample in the original dataset, `genotype_1` and `genotype_2`, the genotype of the first and second duplicated samples for the current marker, respectively.
7. One result file is created:
   - `dup_samples.summary`: the completion and summarized concordance of each duplicated sample pair. The first two columns (`origIndex` and `dupIndex` are the indexes of the duplicated sample in the original and duplicated transposed pedfiles, respectively.
8. One result file is created
   - `dup_samples.concordance`: the pairwise concordance of each duplicated samples.
9. One set of PLINK's transposed pedfiles:
   - `dup_samples.duplicated_samples`: the dataset containing the duplicated samples from the original dataset.
10. Two output files are created:
    - `dup_samples.chosen_samples.info`: a list of samples that were chosen for completion according to their completion and summarized concordance with their duplicates. Again, their indexes in the original and duplicated transposed pedfiles are saved (the two first columns).
    - `dup_samples.excluded_samples.info`: a list of samples that were not chosen for completion according to their completion and summarized concordance with their duplicates.
11. Multiple output files are created, along with on set of PLINK's transposed pedfiles:
    - `dup_samples.zeroed_out`: the list of genotypes that were zeroed out during completion of the chosen duplicated samples.
    - `dup_samples.not_good_enough`: the list of samples that were not good enough (according to completion and concordance) to create the composite sample (the chosen duplicated samples).

12. Two sets of PLINK's transposed pedfiles are created:
    - `dup_samples.chosen_samples`: a transposed pedfiles containing the completed chosen samples.
    - `dup_samples.final`: the final dataset.

5.3.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.DupSamples.duplicated_samples*

5.4 Duplicated Markers Module

The usage of the standalone module is shown below:

```
$ pyGenClean_duplicated_snps --help
usage: pyGenClean_duplicated_snps [-h] [-v] --tfile FILE
                                  [--snp-completion-threshold FLOAT]
                                  [--snp-concordance-threshold FLOAT]
                                  [--frequency_difference FLOAT] [--out FILE]

Extracts and merges duplicated markers.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --tfile FILE          The input file prefix (will find the tped and tfam
                        file by appending the prefix to .tped and .tfam,
                        respectively. A .map file is also required.

Options:
  --snp-completion-threshold FLOAT
                        The completion threshold to consider a replicate when
                        choosing the best replicates and for composite
                        creation. [default: 0.9]
  --snp-concordance-threshold FLOAT
                        The concordance threshold to consider a replicate when
                        choosing the best replicates and for composite
                        creation. [default: 0.98]
  --frequency_difference FLOAT
                        The maximum difference in frequency between duplicated
                        markers [default: 0.05]

Output File:
  --out FILE            The prefix of the output files. [default: dup_snps]
```

5.4.1 Input Files

This module uses PLINK's transposed pedfile format (`tped` and `tfam` files). It also requires a `map` file to speed up the process of finding the duplicated markers, so that the `tped` file is not read.

5.4.2 Procedure

Here are the steps performed by the module:

1. Reads the `map` file to gather marker's position.
2. Reads the `tfam` file.
3. Finds the unique markers using the `map` file.
4. Process the `tped` file, finding unique and duplicated markers according to chromosomal positions.
5. If there are no duplicated markers, stop here.
6. If there are duplicated markers, print a `tped` and `tfam` file containing the duplicated markers.
7. Computes the frequency of the duplicated markers (using Plink) and read the output file.
8. Computes the concordance and pairwise completion of each of the duplicated markers.
9. Prints the problematic duplicated markers with a file containing the summary of the statistics (completion and pairwise concordance).
10. Print the pairwise concordance in a file (matrices).
11. Choose the best duplicated markers using concordance and completion.
12. Completes the chosen markers with the remaining duplicated markers.
13. Creates the final `tped` file, containing the unique markers, the chosen duplicated markers that were completed, and the problematic duplicated markers (for further analysis). This set excludes markers that were used for completing the chosen ones.

5.4.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* dup_snps]):

1. If the marker names are not unique, one file is created:
    - `dup_snps.duplicated_marker_names`: a list of marker names and chromosomal positions for each marker with duplicated names. This file is not created if there are no markers with duplicated names.
2. No files are created.
3. No files are created.
4. One set of transposed pedfiles.
    - `dup_snps.unique_snps`: the transposed pedfiles containing the unique markers (according to chromosomal positions).
5. If there are no duplicated markers (according to chromosomal positions), the transposed pedfiles created at the previous step are copied to a new set of transposed pedfiles.
    - `dup_snps.final`: the final transposed pedfiles.
6. One set of transposed pedfiles.
    - `dup_snps.duplicated_snps`: the transposed pedfiles containing the duplicated markers (according to chromosomal positions).
7. One set of PLINK's result file.

- dup_snps.duplicated_snps: the file with the frq extension contains the frequency of each duplicated markers.

8. No files are created.
9. Multiple files are created.
   - dup_snps.summary: contains the completion and pairwise concordance between duplicated markers.
   - dup_snps.problems: contains the list of markers with "problems" that can't be used for further completion of the duplicated markers. (either a difference in MAF [diff_frequency], a difference in the minor allele [diff_minor_allele], two homozygous markers where one is flipped [homo_flip], markers with flipped alleles [flip], one marker is homozygous, the other is heterozygous [homo_hetero], one marker is homozygous, the other is heterozygous but one is flipped [homo_hetero_flip] or any other problem [problem].
10. One output file is created.
    - dup_snps.concordance: a matrix containing a pairwise concordance comparison for each duplicated markers.
11. Two output files are created.
    - dup_snps.chosen_snps.info: the list of duplicated markers that were chosen for completion with the other markers (the best of the duplicated markers, according to concordance and completion).
    - dup_snps.not_chosen_snps.info: the list of duplicated markers that were not chosen for completion with the other markers.
12. Multiple output files are created along with a set of transposed pedfiles.
    - dup_snps.zeroed_out: the list of genotypes that were zeroed out while completing the chosen duplicated markers with the others. Each line contains the id of the sample and the name of the marker that was zeroed out.
    - dup_snps.not_good_enough: the list of markers that were not good enough (according to concordance and completion) to complete the best of the duplicated markers.
    - dup_snps.removed_duplicates: the list of markers that were used to complete the chosen duplicated markers. Those markers were removed from the dataset.
    - dup_snps.chosen_snps: the transposed pedfiles containing the completed chosen duplicated markers (a composite of all the duplicated markers that were good enough).
13. On set of transposed pedfiles.
    - dup_snps.final: the final dataset, containing the unique markers, the chosen duplicated markers that were complete (composite) and the duplicated markers that weren't completed because of various problems.

5.4.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.DupSNPs.duplicated_snps*

5.5 Clean No Call and Only Heterozygous Markers Module

The usage of the standalone module is shown below:

```
$ pyGenClean_clean_noCall_hetero_snps --help
usage: pyGenClean_clean_noCall_hetero_snps [-h] [-v] --tfile FILE [--out FILE]

Removes "no calls" only and heterozygous only markers.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --tfile FILE   The input file prefix (will find the tped and tfam file by
                 appending the prefix to .tped and .tfam, respectively.

Output File:
  --out FILE     The prefix of the output files. [default:
                 clean_noCall_hetero]
```

5.5.1 Input Files

This module uses the transposed pedfile format separated by tabulations (`tped` and `tfam` files) for the source data set (the data of interest).

5.5.2 Procedure

Here are the steps performed by the module:

1. Reads the transposed pedfiles and extract markers which are all heterozygous or all failed from the dataset.

5.5.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_noCall_hetero_snps`]):

1. One transposed pedfiles and two custom output files are created:
   - `clean_noCall_hetero`: the transposed pedfiles separated by tabulations containing the new dataset, with markers which are all heterozygous or all failed were removed from the initial dataset.
   - `clean_noCall_hetero.allHetero`: the list of markers which were all heterozygous in the initial dataset.
   - `clean_noCall_hetero.allFailed`: the list of markers which were all failed in the initial dataset.

5.5.4 The Plots

A standalone script has been created so that heterozygosity rates can be visualized using histograms or box plots. This script has not yet been included in the automated pipeline, so it needs to be started manually.

```
$ pyGenClean_heterozygosity_plot --help
usage: pyGenClean_heterozygosity_plot [-h] [-v] --tfile FILE [--boxplot]
                                      [--format FORMAT] [--bins INT]
                                      [--xlim FLOAT FLOAT] [--ymax FLOAT]
                                      [--out FILE]
```

```
Plots the distribution of the heterozygosity ratio.

optional arguments:
  -h, --help           show this help message and exit
  -v, --version        show program's version number and exit

Input File:
  --tfile FILE         The prefix of the transposed file

Options:
  --boxplot            Draw a boxplot instead of a histogram.
  --format FORMAT      The output file format (png, ps, pdf, or X11 formats are
                       available). [default: png]
  --bins INT           The number of bins for the histogram. [default: 100]
  --xlim FLOAT FLOAT   The limit of the x axis (floats).
  --ymax FLOAT         The maximal Y value.

Output File:
  --out FILE           The prefix of the output files. [default:
                       heterozygosity]
```

The script produces either a histogram (see the *Heterozygosity rate histogram* figure) or a box plot (see the *Heterozygosity rate box plot* figure) of samples' heterozygosity rates.

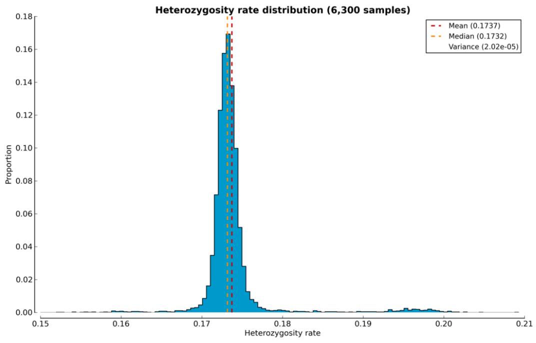

Fig. 5.1: Heterozygosity rate histogram

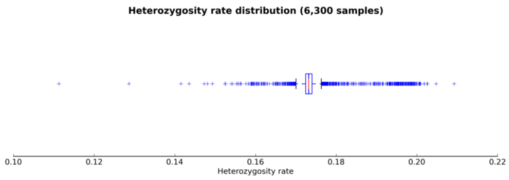

Fig. 5.2: Heterozygosity rate box plot

5.5.5 The Algorithm

For more information about the actual algorithms and source codes, refer to the following pages.

- *pyGenClean.NoCallHetero.clean_noCall_hetero_snps*

- *pyGenClean.NoCallHetero.heterozygosity_plot*

5.6 Sample Missingness Module

The usage of the standalone module is shown below:

```
$ pyGenClean_sample_missingness --help
usage: pyGenClean_sample_missingness [-h] [-v] --ifile FILE [--is-bfile]
                                     [--mind FLOAT] [--out FILE]

Computes sample missingness using Plink.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --ifile FILE   The input file prefix (by default, this input file must be a
                 tfile. If options --is-bfile is used, the input file must be
                 a bfile).

Options:
  --is-bfile     The input file (--ifile) is a bfile instead of a tfile.
  --mind FLOAT   The missingness threshold (remove samples with more than x
                 percent missing genotypes). [Default: 0.100]

Output File:
  --out FILE     The prefix of the output files (wich will be a Plink binary
                 file). [default: clean_mind]
```

5.6.1 Input Files

This module uses either PLINK's binary file format (`bed`, `bim` and `fam` files) or the transposed pedfile format separated by tabulations (`tped` and `tfam`) for the source data set (the data of interest).

5.6.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to remove samples with a high missing rate (above a user defined threshold).

5.6.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_geno`]):

1. One set of PLINK's output and result files:

    - `clean_mind`: the new dataset with samples having a high missing rate removed (above a user defined threshold). The file `clean_mind.irem` contains a list of samples that were removed.

5.6.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.SampleMissingness.sample_missingness*

5.7 Marker Missingness Module

The usage of the standalone module is shown below:

```
$ pyGenClean_snp_missingness --help
usage: pyGenClean_snp_missingness [-h] [-v] --bfile FILE [--geno FLOAT]
                                  [--out FILE]

Computes marker missingness using Plink.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --bfile FILE   The input file prefix (will find the plink binary files by
                 appending the prefix to the .bim, .bed and .fam files,
                 respectively.

Options:
  --geno FLOAT   The missingness threshold (remove SNPs with more than x
                 percent missing genotypes). [Default: 0.020]

Output File:
  --out FILE     The prefix of the output files. [default: clean_geno]
```

5.7.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.7.2 Procedure

Here are the steps performed by the module:

1. Runs Plink to remove markers with a missing rate above a user defined threshold.
2. Finds the markers that were removed (those that have a missing rate above the user defined threshold.

5.7.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_geno`]):

1. One set of Plink output files:
    - `clean_geno.fam`: the dataset with markers having a high missing rate removed (according to a user defined threshold).

2. One custom file:

   - `clean_geno.removed_snps`: a list of markers that have a high missing rate (above a user defined threshold).

5.7.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.MarkerMissingness.snp_missingness*

5.8 Sex Check Module

The usage of the standalone module is shown below:

```
$ pyGenClean_sex_check --help
usage: pyGenClean_sex_check [-h] [-v] --bfile FILE [--femaleF FLOAT]
                            [--maleF FLOAT] [--nbChr23 INT] [--gender-plot]
                            [--sex-chr-intensities FILE]
                            [--gender-plot-format FORMAT] [--lrr-baf]
                            [--lrr-baf-raw-dir DIR] [--lrr-baf-format FORMAT]
                            [--out FILE]

Check sample's gender using Plink.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the Plink binary
                        files by appending the prefix to the .bed, .bim, and
                        .fam files, respectively.

Options:
  --femaleF FLOAT       The female F threshold. [default: < 0.300000]
  --maleF FLOAT         The male F threshold. [default: > 0.700000]
  --nbChr23 INT         The minimum number of markers on chromosome 23 before
                        computing Plink's sex check [default: 50]

Gender Plot:
  --gender-plot         Create the gender plot (summarized chr Y intensities
                        in function of summarized chr X intensities) for
                        problematic samples.
  --sex-chr-intensities FILE
                        A file containing alleles intensities for each of the
                        markers located on the X and Y chromosome for the
                        gender plot.
  --gender-plot-format FORMAT
                        The output file format for the gender plot (png, ps,
                        pdf, or X11 formats are available). [default: png]

LRR and BAF Plot:
  --lrr-baf             Create the LRR and BAF plot for problematic samples
  --lrr-baf-raw-dir DIR
                        Directory or list of directories containing
                        information about every samples (BAF and LRR).
  --lrr-baf-format FORMAT
                        The output file format for the LRR and BAF plot (png,
                        ps, pdf, or X11 formats are available). [default: png]

Output File:
  --out FILE            The prefix of the output files (which will be a Plink
                        binary file. [default: sexcheck]
```

5.8.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

If the option of generating the gender plot is used, a file containing intensities information about each markers on the sexual chromosomes is required. This file (which could be gzipped) should contain at least the following columns:

- `Sample ID`: the unique sample id for each individual.
- `SNP Name`: the unique name of each markers.
- `Chr`: the name of the chromosome on which each marker is located.
- `X`: the intensities of the first allele of each marker.
- `Y`: the intensities of the second allele of each marker.

If the options of generating the BAF and LRR values is used, the name of a directory containing intensities file for each sample is required. The name of each file should be the unique sample id. This file (which could be gzipped) should contain at least the following columns:

- `Chr`: the name of the chromosome on which each marker is located.
- `Position`: the position of the marker on the chromosome.
- `B Allele Freq`: the BAF value of each marker.
- `Log R Ratio`: the LRR value of each marker.

If the two plotting module is used alone, one more file is required per module: a list of samples with gender problem and explanation for *`pyGenClean.SexCheck.gender_plot`*, and only the list of samples with gender problem for *`pyGenClean.SexCheck.baf_lrr_plot`* (both files are provided by the *`pyGenClean.SexCheck.sex_check`* module).

5.8.2 Procedure

Here are the steps performed by the module:

1. Checks if there are enough markers on the chromosome 23. If not, the module stops here.
2. Runs the sex check analysis using Plink.
3. If there are no sex problems, the module quits.
4. Creates the recoded file for the chromosome 23.
5. Computes the heterozygosity percentage on the chromosome 23.
6. If there are enough markers on chromosome 24 (at least 1), creates the recoded file for this chromosome.
7. Computes the number of no call on the chromosome 24.
8. If required, plots the gender plot.

   (a) If there are `summarized_intensities` provided, reads the files and skips to step vi.
   (b) Reads the `bim` file to get markers on the sexual chromosomes.
   (c) Reads the `fam` file to get individual's gender.
   (d) Reads the file containing samples with sex problems.
   (e) Reads the intensities and summarizes them.
   (f) Plots the summarized intensities.

9. If required, plots the BAF and LRR plots.
   (a) Reads the problematic samples.
   (b) Finds and checks the raw files for each of the problematic samples.
   (c) Plots the BAF and LRR plots.

5.8.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e* `sexcheck`]):

1. No output files created.
2. One set of PLINK's result files:
   - `sexcheck`: the result of the sex check procedure from Plink.
3. Two files are created if there are sex problems:
   - `sexcheck.list_problem_sex`: a summary of samples with sex problem.
   - `sexcheck.list_problem_sex_ids`: the list of sample ids with sex problem.
4. One set of Plink's files:
   - `sexcheck.chr23_recodeA`: the recoded file for the chromosome 23.
5. One custom output file:
   - `sexcheck.chr23_recodeA.raw.hetero`: the heterozygosity percentage on the chromosome 23. The file includes the following columns: `PED` (the pedigree ID), `ID` (the individual ID), `SEX` (the gender) and `HETERO` (the heterozygosity).
6. One set of Plink's files:
   - `sexcheck.chr24_recodeA`: the recoded file for the chromosome 24.
7. One custom output file:
   - `sexcheck.chr24_recodeA.raw.noCall`: the number of no call on the chromosome 24. The file includes the following columns: `PED` (the pedigree ID), `ID` (the individual ID), `SEX` (the gender), `nbGeno` (the number of genotypes on the chromosome 24) and `nbNoCall` (the number of genotypes that were not called on chromosome 24).
8. Multiple files and one plot. The files are created so that the plot can be generated again with different parameters (since the summarized intensities for each sample are really long to compute).
   - `sexcheck.png`: the gender plot (see Figure *Gender plot*).
   - `sexcheck.ok_females.txt`: the list of females without sex problem, including their summarized intensities on chromosome 23 and 24.

- `sexcheck.ok_males.txt`: the list of males without sex problem, including their summarized intensities on chromosome 23 and 24.
- `sexcheck.ok_unknowns.txt`: the list of unknown gender without sex problem, including their summarized intensities on chromosome 23 and 24.
- `sexcheck.problematic_females.txt`: the list of females with sex problem, including their summarized intensities on chromosome 23 and 24.
- `sexcheck.problematic_males.txt`: the list of males with sex problem, including their summarized intensities on chromosome 23 and 24.
- `sexcheck.problematic_unknowns.txt`: the list of unknown gender with sex problem, including their summarized intensities on chromosome 23 and 24. When this file is created by the *`pyGenClean.SexCheck.sex_check`* module, it is empty.

9. A directory containing one file per individual with gender problem (see Figure *BAF and LRR plot*).

5.8.4 The Plots

The plot generated by the *`pyGenClean.SexCheck.gender_plot`* module (the *Gender plot* figure) represents the summarized intensities of each sample of the data set. The color code represent the gender of each sample. Blue and red dots represent the males and females without gender problem, respectively. The green and purple triangles represent the males and females with gender problem. The gray dots and triangles represent the samples with unknown gender, with and without gender problem, respectively. This plot makes possible to find samples with sexual chromosomes abnormalities, such as males which are XXY or females with only one copy of the X chromosome (X0). Males that appear to be females and vice versa might in fact be sample mix up and would require further analysis.

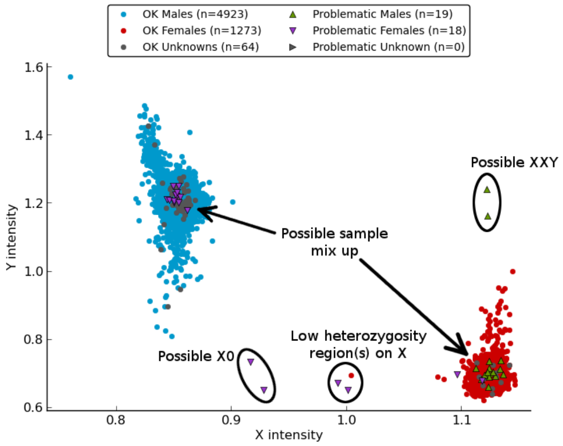

Fig. 5.3: Gender plot

The *Gender plot* figure can also be manually created after the data clean up pipeline, using its results and this following standalone script:

```
$ pyGenClean_gender_plot --help
usage: pyGenClean_gender_plot [-h] [-v] --bfile FILE [--intensities FILE]
                              [--summarized-intensities FILE] --sex-problems
                              FILE [--format FORMAT] [--xlabel STRING]
                              [--ylabel STRING] [--out FILE]

Plots the gender using X and Y chromosomes' intensities
```

```
optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The plink binary file containing information about
                        markers and individuals. Must be specified if
                        '--summarized-intensities' is not.
  --intensities FILE    A file containing alleles intensities for each of the
                        markers located on the X and Y chromosome. Must be
                        specified if '--summarized-intensities' is not.
  --summarized-intensities FILE
                        The prefix of six files (prefix.ok_females.txt,
                        prefix.ok_males.txt, prefix.ok_unknowns.txt,
                        problematic_females.txt, problematic_males.txt and
                        problematic_unknowns.txt) containing summarized chr23
                        and chr24 intensities. Must be specified if '--bfile'
                        and '--intensities' are not.
  --sex-problems FILE   The file containing individuals with sex problems.
                        This file is not read if the option 'summarized-
                        intensities' is used.

Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --xlabel STRING       The label of the X axis. [default: X intensity]
  --ylabel STRING       The label of the Y axis. [default: Y intensity]

Output File:
  --out FILE            The prefix of the output files (which will be a Plink
                        binary file. [default: sexcheck]
```

The log R ratio (LRR) for a sample is the log ratio of the normalized R value for the marker divided by the expected normalized R value. Hence, a value of 0 means 2 copies. A drop in the LRR shows a loss of a copy, while an increasing LRR shows a gain of a copy. Expected LRR values on chromosome 23 for female and female are 0 and [-0.5, -1], respectively. The LRR values of each markers on both the X and Y chromosomes are shown in the *BAF and LRR plot* figure.

The B allele frequency (BAF) for a sample shows the theta value for a marker, corrected for cluster position. For a normal number of copies, markers should have a BAF around 1 (homozygous for the B allele), 0.5 (heterozygous) or 0 (homozygous for A allele). Normal females should have the three lines across the chromosome. Normal males should only have two lines, located near 1 or 0. The BAF values of each markers on both the X and Y chromosomes are shown in the *BAF and LRR plot* figure.

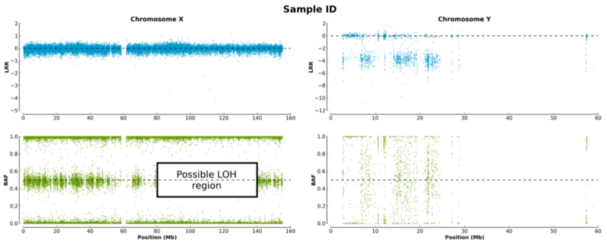

Fig. 5.4: BAF and LRR plot

The *BAF and LRR plot* figure can also be manually created after the data clean up pipeline, using this following standalone script:

```
$ pyGenClean_baf_lrr_plot --help
usage: pyGenClean_baf_lrr_plot [-h] [-v] --problematic-samples FILE
                               [--use-full-ids] [--full-ids-delimiter CHAR]
                               --raw-dir DIR [--format FORMAT] [--out FILE]

Plots the BAF and LRR of problematic samples.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --problematic-samples FILE
                        A file containing the list of samples with sex
                        problems (family and individual ID required, separated
                        by a single tabulation). Uses only the individual ID
                        by default, unless --use-full-ids is used.
  --use-full-ids        Use this options to use full sample IDs (famID and
                        indID). Otherwise, only the indID will be use.
  --full-ids-delimiter CHAR
                        The delimiter between famID and indID for the raw file
                        names. [default: _]
  --raw-dir DIR         Directory containing information about every samples
                        (BAF and LRR).

Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]

Output File:
  --out FILE            The prefix of the output files. [default: sexcheck]
```

5.8.5 The Algorithm

For more information about the actual algorithms and source codes, refer to the following pages.

- *pyGenClean.SexCheck.sex_check*
- *pyGenClean.SexCheck.gender_plot*
- *pyGenClean.SexCheck.baf_lrr_plot*

5.9 Plate Bias Module

The usage of the standalone module is shown below:

```
$ pyGenClean_plate_bias --help
usage: pyGenClean_plate_bias [-h] [-v] --bfile FILE --loop-assoc FILE
                             [--pfilter FLOAT] [--out FILE]

Checks for plate bias.

optional arguments:
  -h, --help          show this help message and exit
  -v, --version       show program's version number and exit

Input File:
  --bfile FILE        The input file prefix (will find the plink binary files
                      by appending the prefix to the .bim, .bed and .fam files,
                      respectively.
  --loop-assoc FILE   The file containing the plate organization of each
                      samples. Must contains three column (with no header):
                      famID, indID and plateName.

Options:
  --pfilter FLOAT     The significance threshold used for the plate effect.
                      [default: 1.0e-07]

Output File:
  --out FILE          The prefix of the output files. [default: plate_bias]
```

5.9.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.9.2 Procedure

Here are the steps performed by the module:

1. Runs the plate bias analysis using Plink.
2. Extracts the list of significant markers after plate bias analysis.
3. Computes the frequency of all significant markers after plate bias analysis.

5.9.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `plate_bias`]):

1. One set of PLINK's result files:
    - `plate_bias`: this includes file like `plate_bias.PLATE_NAME.assoc.fisher` containing a list of markers that were significant after the plate bias analysis.
2. One file:
    - `plate_bias.significant_SNPs.txt`: a file containing a list of markers that were significant after the plate bias analysis (contained in all the `*.fisher` files).
3. One set of PLINK's result files:
    - `plate_bias.significant_SNPs`: the result files containing the frequency of all the markers that were significant after the plate bias analysis (contained in all the `*.fisher` files).

5.9.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.PlateBias.plate_bias*

5.10 Heterozygous Haploid Module

The usage of the standalone module is shown below:

```
$ pyGenClean_remove_heterozygous_haploid --help
Usage: pyGenClean_remove_heterozygous_haploid [-h] [-v] --bfile FILE
                                              [--out FILE]

Removes heterozygous haploid genotypes.

Optional arguments:
  -h, --help      show this help message and exit
  -v, --version   show program's version number and exit

Input File:
  --bfile FILE    The input file prefix (will find the plink binary files by
                  appending the prefix to the .bim, .bed and .fam files,
                  respectively.

Output File:
  --out FILE      The prefix of the output files. [default:
                  without_hh_genotypes]
```

5.10.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.10.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to remove the heterozygous haploid genotypes.

5.10.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `without_hh_genotypes`]):

1. On set of Plink's output files:
    - `without_hh_genotypes`: the data set with heterozygous haploid genotypes removed.

5.10.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.HeteroHap.remove_heterozygous_haploid*

5.11 Related Samples Module

The usage of the standalone module is shown below:

```
$ pyGenClean_find_related_samples --help
usage: pyGenClean_find_related_samples [-h] [-v] --bfile FILE [--genome-only]
                                       [--min-nb-snp INT]
                                       [--indep-pairwise STR STR STR]
                                       [--maf FLOAT] [--ibs2-ratio FLOAT]
                                       [--sge] [--sge-walltime TIME]
                                       [--sge-nodes INT INT]
                                       [--line-per-file-for-sge INT]
                                       [--out FILE]

Finds related samples according to IBS values.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.)

Options:
  --genome-only         Only create the genome file
  --min-nb-snp INT      The minimum number of markers needed to compute IBS
                        values. [Default: 10000]
  --indep-pairwise STR STR STR
                        Three numbers: window size, window shift and the r2
                        threshold. [default: ['50', '5', '0.1']]
  --maf FLOAT           Restrict to SNPs with MAF >= threshold. [default:
                        0.05]
  --ibs2-ratio FLOAT    The initial IBS2* ratio (the minimum value to show in
                        the plot. [default: 0.8]
  --sge                 Use SGE for parallelization.
  --sge-walltime TIME   The walltime for the job to run on the cluster. Do not
                        use if you are not required to specify a walltime for
                        your jobs on your cluster (e.g. 'qsub
                        -lwalltime=1:0:0' on the cluster).
  --sge-nodes INT INT   The number of nodes and the number of processor per
                        nodes to use (e.g. 'qsub -lnodes=X:ppn=Y' on the
                        cluster, where X is the number of nodes and Y is the
                        number of processor to use. Do not use if you are not
                        required to specify the number of nodes for your jobs
                        on the cluster.
  --line-per-file-for-sge INT
                        The number of line per file for SGE task array.
                        [default: 100]

Output File:
  --out FILE            The prefix of the output files. [default: ibs]
```

5.11.1 Input Files

This module uses PLINK's binary file format (bed, bim and fam files) for the source data set (the data of interest).

5.11.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to extract markers according to LD.
2. Checks if there is enough markers after pruning.
3. Extract markers according to LD.
4. Runs Plink with the genome option to compute the IBS values.
5. Finds related individuals and gets values for plotting.
6. Plots Z1 in function of IBS2 ratio for related individuals.
7. Plots Z2 in function of IBS2 ratio for related individuals.

5.11.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* ibs]):

1. One set of PLINK's result files:
   - ibs.pruning_0.1: the results of the pruning process of Plink. The value depends on the option of --indep-pairwise. The markers that are kept are in the file ibs.pruning_0.1.prune.in.
2. No file created.
3. One set of PLINK's binary files:
   - ibs.pruned_data: the data sets containing only the marker from the first step (the list is in ibs.pruning_0.1.prune.in).
4. One set of PLINK's result files (two if --sge is used):
   - ibs.frequency: PLINK's result files when computing the frequency of each of the pruned markers. This data set will exist only if the option --sge is used.
   - ibs.genome: PLINK's results including IBS values.
5. One file provided by the *pyGenClean.RelatedSamples.find_related_samples* and three files provided by *pyGenClean.RelatedSamples.merge_related_samples*:
   - **ibs.related_individuals: a subset of the ibs.genome.genome file** containing only samples that are considered to be related. Three columns are appended to the original ibs.genome.genome file: IBS2_ratio (the value that is considered to find related individuals), status (the type of relatedness [*e.g.* twins]) and code (a numerical code that represent the status). This file is provided by the *pyGenClean.RelatedSamples.find_related_samples* module.
   - ibs.merged_related_individuals: a file aggregating related samples in groups, containing the following columns: index (the group number), FID1 (the family ID of the first sample), IID1 (the individual ID of the first sample), FID2 (the family ID of the second sample), IID2 (the individual ID of the second sample) and status (the type of relatedness between the two samples). This file is provided by the merge_related_samples.

   - `ibs.chosen_related_individuals`: the related individuals that were randomly chosen from each group to be kept in the final data set. This file is provided by the `merge_related_samples`.
   - `ibs.discarded_related_individual`: the related individuals that needs to be discarded, so that the final data set include only unrelated individuals. This file is provided by the `merge_related_samples`.

6. One image file:
   - `ibs.related_individuals_z1.png`: a plot showing the Z_1 value in function of the $IBS2^*_{ratio}$ for all samples above a certain $IBS2^*_{ratio}$ (the default threshold is 0.8). See Figure *Z1 in function of IBS2 ratio*.
7. One image file:
   - `ibs.related_individuals_z2.png`: a plot showing the Z_2 value in function of the $IBS2^*_{ratio}$ for all samples above a certain $IBS2^*_{ratio}$ (the default threshold is 0.8). See Figure *Z2 in function of IBS2 ratio*.

5.11.4 The Plots

The first plot (*Z1 in function of IBS2 ratio* figure) that is created is Z_1 in function of $IBS2^*_{ratio}$ (where each point represents a pair of related individuals. The color code comes from the different value of Z_0, Z_1 and Z_2, as described in the *`pyGenClean.RelatedSamples.find_related_samples.extractRelatedIndividuals()`* function. In this plot, there are four locations where related samples tend to accumulate (first degree relatives (full sibs), second degree relatives (half-sibs, grand-parent-child or uncle-nephew), parent-child and twins (or duplicated samples). The unknown sample pairs represent possible undetected related individuals.

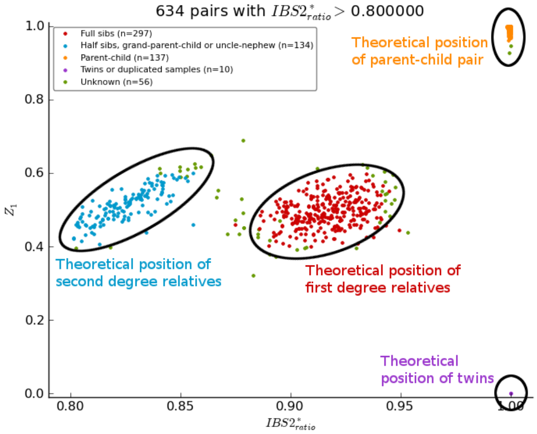

Fig. 5.5: Z1 in function of IBS2 ratio

The second plot (*Z2 in function of IBS2 ratio* figure) that is created is Z_2 in function of $IBS2^*_{ratio}$ (where each point represents a pair of related individuals. It's just another representation of relatedness of sample pairs, where the location of the "clusters" is different.

5.11.5 Finding Outliers

A standalone script was created in order to regroup related samples in different subset. The usage is as follow:

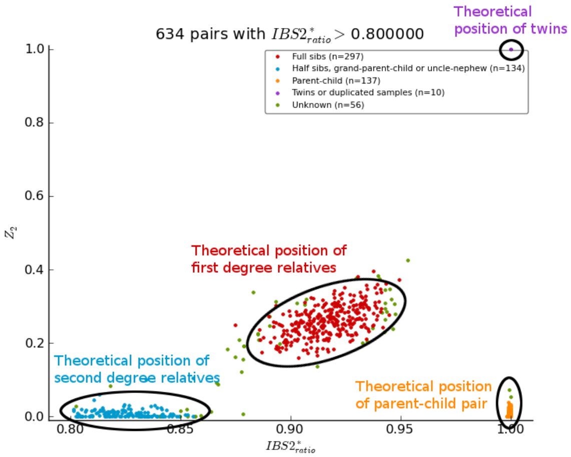

Fig. 5.6: Z2 in function of IBS2 ratio

```
$ pyGenClean_merge_related_samples --help
usage: pyGenClean_merge_related_samples [-h] [-v] --ibs-related FILE
                                        [--no-status] [--out FILE]

Merges related samples according to IBS.

optional arguments:
  -h, --help           show this help message and exit
  -v, --version        show program's version number and exit

Input File:
  --ibs-related FILE   The input file containing related individuals according
                       to IBS value.

Options:
  --no-status          The input file doesn't have a 'status' column.

Output File:
  --out FILE           The prefix of the output files. [default: ibs_merged]
```

At the end of the analysis, two files are created. The file `*.chosen_related_individuals"` contains a list of randomly selected samples according to their relatedness (to keep only on sample for a group of related samples). The file `*.discarded_related_individuals` contains a list of sample to exclude to only keep unrelated samples in a dataset.

5.11.6 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.RelatedSamples.find_related_samples*
- *pyGenClean.RelatedSamples.merge_related_samples*

5.12 Ethnicity Module

The usage of the standalone module is shown below:

```
$ pyGenClean_check_ethnicity --help
usage: pyGenClean_check_ethnicity [-h] [-v] --bfile FILE [--skip-ref-pops]
                                  [--ceu-bfile FILE] [--yri-bfile FILE]
                                  [--jpt-chb-bfile FILE] [--min-nb-snp INT]
                                  [--indep-pairwise STR STR STR] [--maf FLOAT]
                                  [--sge] [--sge-walltime TIME]
                                  [--sge-nodes INT INT]
                                  [--ibs-sge-walltime TIME]
                                  [--ibs-sge-nodes INT INT]
                                  [--line-per-file-for-sge INT]
                                  [--nb-components INT] [--outliers-of POP]
                                  [--multiplier FLOAT] [--xaxis COMPONENT]
                                  [--yaxis COMPONENT] [--format FORMAT]
                                  [--title STRING] [--xlabel STRING]
                                  [--ylabel STRING] [--create-scree-plot]
                                  [--scree-plot-title TITLE] [--out FILE]

Checks samples' ethnicity using reference populations and IBS.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.
  --skip-ref-pops       Perform the MDS computation, but skip the three
                        reference panels.
  --ceu-bfile FILE      The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.) for the CEU population
  --yri-bfile FILE      The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.) for the YRI population
  --jpt-chb-bfile FILE  The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.) for the JPT-CHB population

Options:
  --min-nb-snp INT      The minimum number of markers needed to compute IBS
                        values. [Default: 8000]
  --indep-pairwise STR STR STR
                        Three numbers: window size, window shift and the r2
                        threshold. [default: ['50', '5', '0.1']]
  --maf FLOAT           Restrict to SNPs with MAF >= threshold. [default:
                        0.05]
  --sge                 Use SGE for parallelization.
  --sge-walltime TIME   The walltime for the job to run on the cluster. Do not
                        use if you are not required to specify a walltime for
                        your jobs on your cluster (e.g. 'qsub
                        -lwalltime=1:0:0' on the cluster).
  --sge-nodes INT INT   The number of nodes and the number of processor per
                        nodes to use (e.g. 'qsub -lnodes=X:ppn=Y' on the
                        cluster, where X is the number of nodes and Y is the
                        number of processor to use. Do not use if you are not
                        required to specify the number of nodes for your jobs
                        on the cluster.
  --ibs-sge-walltime TIME
                        The walltime for the IBS jobs to run on the cluster.
                        Do not use if you are not required to specify a
                        walltime for your jobs on your cluster (e.g. 'qsub
                        -lwalltime=1:0:0' on the cluster).
  --ibs-sge-nodes INT INT
                        The number of nodes and the number of processor per
                        nodes to use for the IBS jobs (e.g. 'qsub
                        -lnodes=X:ppn=Y' on the cluster, where X is the number
                        of nodes and Y is the number of processor to use. Do
                        not use if you are not required to specify the number
                        of nodes for your jobs on the cluster.
  --line-per-file-for-sge INT
                        The number of line per file for SGE task array for the
                        IBS jobs. [default: 100]
  --nb-components INT   The number of component to compute. [default: 10]

Outlier Options:
  --outliers-of POP     Finds the outliers of this population. [default: CEU]
  --multiplier FLOAT    To find the outliers, we look for more than x times
                        the cluster standard deviation. [default: 1.9]
  --xaxis COMPONENT     The component to use for the X axis. [default: C1]
  --yaxis COMPONENT     The component to use for the Y axis. [default: C2]

MDS Plot Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --title STRING        The title of the MDS plot. [default: C2 in function of
                        C1 - MDS]
  --xlabel STRING       The label of the X axis. [default: C1]
  --ylabel STRING       The label of the Y axis. [default: C2]

Scree Plot Options:
  --create-scree-plot   Computes Eigenvalues and creates a scree plot.
  --scree-plot-title TITLE
                        The main title of the scree plot [EIGENSOFT results]

Output File:
  --out FILE            The prefix of the output files. [default: ethnicity]
```

5.12.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest) and three sets of binary files for the reference panels (CEU, YRI and JPG-CHB).

5.12.2 Procedure

Here are the steps performed by the module:

1. Finds the overlapping markers between the three reference panels and the source panel.
2. Extracts the required markers from all the data sets (source and reference panels).

3. Combines the three reference panels together to create a single data set.
4. Renames the reference panel's markers so that they match the names of the markers in the source panel.
5. Computes the frequency of all the markers from the reference and the source panels.
6. Finds the markers to flip in the reference panel, to enable fast comparison with the source panel.
7. Excludes markers that cannot be flip from the reference and the source panels.
8. Flips the markers that need to be in the reference panel.
9. Combines the reference and the source panels.
10. Computes the IBS values (see Section *Related Samples Module* for more information).
11. Creates the MDS file from the combined data set and the IBS values.
12. Creates a population file for plotting purposes.
13. Plots the MDS values.
14. Finds the outliers of a given reference population (either CEU, YRI or JPT-CHB).
    (a) Reads the population file.
    (b) Reads the MDS values.
    (c) Computes the three reference population clusters' center.
    (d) Computes three clusters according to the reference population clusters' centers, and finds the outliers of a given reference population.
    (e) Writes the outliers in a file.

5.12.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e* `ethnicity`]):

1. Three files are created:
    - `ethnicity.ref_snp_to_extract`: a list of markers to extract from the reference panels.
    - `ethnicity.source_snp_to_extract`: a list of markers to extract from the source panel.
    - `ethnicity.update_names`: the updated names of the marker in the reference panels, so that they match with the names in the source panel.
2. Four sets of PLINK's binary files are created:
    - `ethnicity.reference_panel.CEU`: the data set containing the extracted markers from the CEU reference population.
    - `ethnicity.reference_panel.YRI`: the data set containing the extracted markers from the YRI reference population.
    - `ethnicity.reference_panel.JPT-CHB`: the data set containing the extracted markers from the JPG-CHB reference population.
    - `ethnicity.source_panel.ALL`: the data set containing the extracted markers from the source population.
3. One required file and one set of PLINK's binary files are created:
    - `ethnicity.reference_panel.ALL.files_to_merge`: the file required by Plink to merge more than two data sets together.

- `ethnicity.reference_panel.ALL`: the data set containing the merged data sets of the three reference population.

4. One set of PLINK's binary files is created:

   - `ethnicity.reference_panel.ALL.rename`: the data set after markers have been renamed in the reference panels.

5. Two sets of PLINK's result files are created:

   - `ethnicity.reference_panel.ALL.rename.frequency`: the frequencies of the markers in the reference panels.
   - `ethnicity.source_panel.ALL.frequency`: the frequencies of the markers in the source panels.

6. Two files are created:

   - `ethnicity.snp_to_flip_in_reference`: the list of markers to flip in the reference panels.
   - `ethnicity.snp_to_remove`: the list of markers to remove because they are not comparable to the markers in the source panel, even after trying to flip them.

7. Two sets of PLINK's binary files are created:

   - `ethnicity.reference_panel.ALL.rename.cleaned`: the data set after the markers found in the previous step are excluded from the reference panels.
   - `ethnicity.source_panel.ALL.cleaned`: the data set after the markers found in the previous step are excluded from the source panel.

8. One set of PLINK's binary files is created:

   - `ethnicity.reference_panel.ALL.rename.cleaned.flipped`: the data set after markers from the reference panels were flipped so that they become comparable with the source panel.

9. One required file and one set of PLINK's binary files are created:

   - `ethnicity.final_dataset_for_genome.files_to_merge`: the file required by Plink to merge more than two data sets together.
   - `ethnicity.final_dataset_for_genome`: the data set containing the merged reference and source panels.

10. Multiple files are created after this step.

    - `ethnicity.ibs` : for more information about those files, see Section *Related Samples Module*.

11. One set of PLINK's result files is created:

    - `ethnicity.mds`: files containing the MDS values.

12. One file is created:

    - `ethnicity.population_file`: the population file required for MDS value plotting.

13. One file is created:

    - `ethnicity.mds.png`: the plot of the MDS values (see Figure *Initial MDS plot*).

14. Four files are created:

    - `ethnicity.before.png`: the MDS values before outliers detection (see Figure *MDS plot before outlier detection*).
    - `ethnicity.after.png`: the MDS values after outliers detection for each of the three reference populations. The shaded points are the outliers (see Figure *MDS plot after outlier detection*).

- `ethnicity.outliers.png`: the MDS values after outliers detection for the selected reference population (default is CEU) (see Figure *Ethnic outliers*).
- `ethnicity.outliers`: the list of outliers (excluding the reference populations).
- `ethnicity.population_file_outliers`: a population file containing the outliers (to help creating a new MDS plot using `pyGenClean.PlinkUtils.plot_MDS_standalone`).

5.12.4 The Plots

Multiple plots are created by this module. The first one (Figure *Initial MDS plot*) is the MDS values right after they are computed by Plink. There is one color per reference populations (CEU in blue, YRI in green and JPT-CHB in purple). The source population is represented as red crosses.

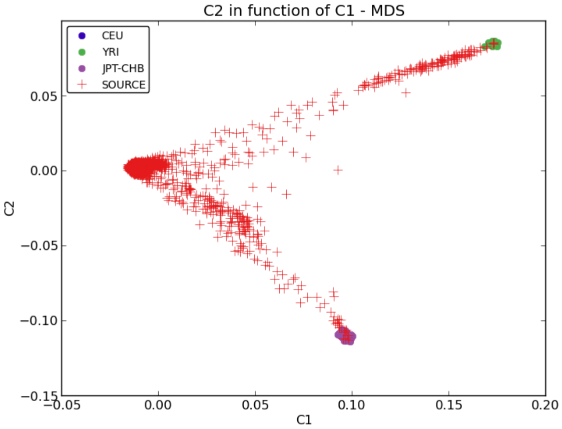

Fig. 5.7: Initial MDS plot

The second one (Figure *MDS plot before outlier detection*) is the MDS values before outlier detection. Points in red, green and blue represent the individuals part of the CEU, YRI and JPT-CHB clusters, respectively. The yellow points represent the center of each of the cluster, when only considering the three reference panels.

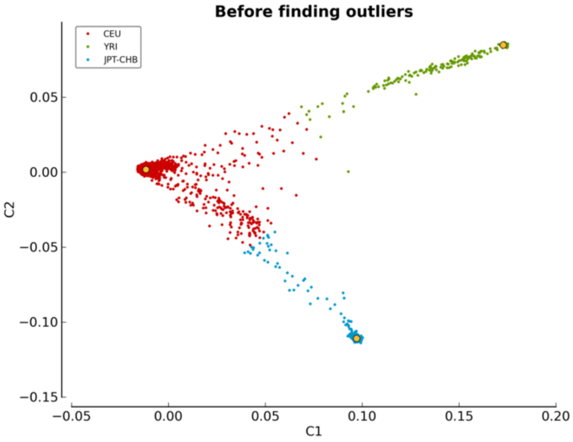

Fig. 5.8: MDS plot before outlier detection

The third plot (Figure *MDS plot after outlier detection*) is the MDS values after outlier detection. Points in red, green and blue represent the individuals part of the CEU, YRI and JPT-CHB clusters, respectively. Outliers are found for each of the three reference populations and they are represented with the same, but lighter color. Once again, the yellow points represent the center of each of the cluster, when only considering the three reference panels.

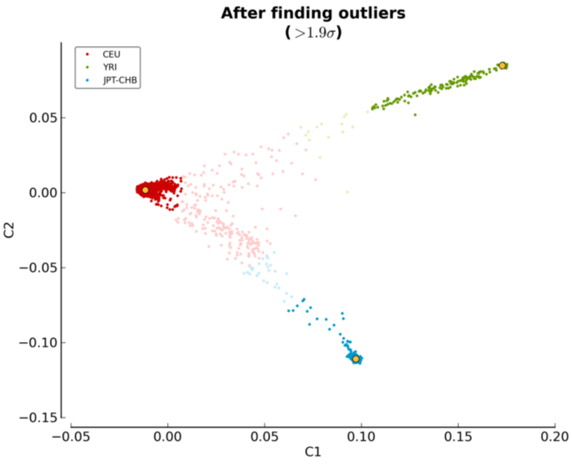

Fig. 5.9: MDS plot after outlier detection

The last plot (Figure *Ethnic outliers*) shows the outliers of the selected reference population (CEU by default). Red, green and blue represent the CEU, YRI and JPT-CHB samples, respectively. Orange represents the individuals from the source panel who are part of the selected reference population. Gray represents the outliers of the selected reference population.

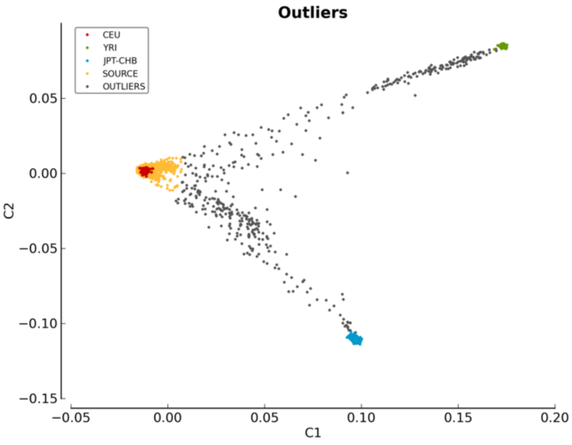

Fig. 5.10: Ethnic outliers

Modifying The Outlier Plot

If you want to manually modify the above figures, have a look at the `pyGenClean.PlinkUtils.plot_MDS_standalone` module. Here is the usage of this script:

```
$ pyGenClean_plot_MDS --help
usage: pyGenClean_plot_MDS [-h] [-v] --file FILE --population-file FORMAT
```

```
                           [--population-order STRING]
                           [--population-colors STRING]
                           [--population-sizes STRING]
                           [--population-markers STRING]
                           [--population-alpha STRING] [--format FORMAT]
                           [--title STRING] [--xaxis STRING] [--xlabel STRING]
                           [--yaxis STRING] [--ylabel STRING]
                           [--legend-position STRING] [--legend-size INT]
                           [--legend-ncol INT] [--legend-alpha FLOAT]
                           [--title-fontsize INT] [--label-fontsize INT]
                           [--axis-fontsize INT] [--adjust-left FLOAT]
                           [--adjust-right FLOAT] [--adjust-top FLOAT]
                           [--adjust-bottom FLOAT] [--out FILE]

Creates a MDS plot.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --file FILE           The MBS file.
  --population-file FORMAT
                        A file containing population information. There must
                        be three columns: famID, indID and population
                        information.

Population Properties:
  --population-order STRING
                        The order to print the different populations.
                        [default: CEU,YRI,JPT-CHB,SOURCE,OUTLIER]
  --population-colors STRING
                        The population point color in the plot [default:
                        377eb8,4daf4a,984ea3,e41a1c,ff7f00]
  --population-sizes STRING
                        The population point size in the plot. [default:
                        12,12,12,8,3]
  --population-markers STRING
                        The population point marker in the plot. [default:
                        .,.,.,+,D]
  --population-alpha STRING
                        The population alpha value in the plot. [default:
                        1.0,1.0,1.0,1.0,1.0]

Graphical Properties:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --title STRING        The title of the MDS plot. [default: C2 in function of
                        C1 - MDS]
  --xaxis STRING        The component to print on the X axis. [default: C1]
  --xlabel STRING       The label of the X axis. [default: C1]
  --yaxis STRING        The component to print on the Y axis. [default: C2]
  --ylabel STRING       The label of the Y axis. [default: C2]
  --legend-position STRING
                        The position of the legend. [default: best]
  --legend-size INT     The size of the legend. [default: 10]
  --legend-ncol INT     The number of column for the legend. [default: 1]
  --legend-alpha FLOAT  The alpha value of the legend frame. [default: 1.0]
  --title-fontsize INT  The font size of the title. [default: 15]
  --label-fontsize INT  The font size of the X and Y labels. [default: 12]
  --axis-fontsize INT   The font size of the X and Y axis. [Default: 12]
  --adjust-left FLOAT   Adjust the left margin. [Default: 0.12]
  --adjust-right FLOAT  Adjust the right margin. [Default: 0.90]
  --adjust-top FLOAT    Adjust the top margin. [Default: 0.90]
  --adjust-bottom FLOAT
                        Adjust the bottom margin. [Default: 0.10]

Output File:
  --out FILE            The prefix of the output files. [default: mds]
```

And here is an example of usage (for a MDS and a population file named `ethnicity.mds.mds` and `ethnicity.population_file_outliers`, respectively), producing the Figure *Ethnic outliers modified*.

```
$ pyGenClean_plot_MDS \
>     --file ethnicity.mds.mds \
>     --population-file ethnicity.population_file_outliers \
>     --population-order SOURCE,CEU,YRI,JPT-CHB,OUTLIER \
>     --population-colors e41a1c,377eb8,4daf4a,984ea3,000000 \
>     --population-markers .,.,.,.,+ \
>     --population-sizes 8,8,8,8,8 \
>     --axis-fontsize 18 \
>     --label-fontsize 18 \
>     --title-fontsize 24 \
>     --adjust-bottom 0.11 \
>     --adjust-left 0.15 \
>     --adjust-right 0.96 \
>     --legend-size 14 \
>     --legend-position lower-right
```

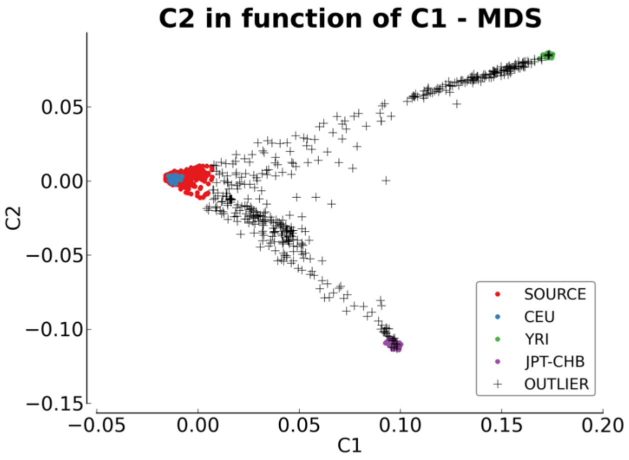

Fig. 5.11: Ethnic outliers modified

5.12.5 Finding Outliers

If the multiplier of the cluster standard deviation was too stringent (or not stringent enough), there is no need to run the module from the start. A standalone script was created for this exact purpose, and it will find the outliers using the `MDS` and population file previously created. Just modify the `--multiplier` option and restart the analysis (which takes about a couple of seconds).

```
$ pyGenClean_find_outliers --help
usage: pyGenClean_find_outliers [-h] [-v] --mds FILE --population-file FILE
                                [--outliers-of POP] [--multiplier FLOAT]
                                [--xaxis COMPONENT] [--yaxis COMPONENT]
                                [--format FORMAT] [--out FILE]

Finds outliers in SOURCE from CEU samples.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

Input File:
  --mds FILE            The MDS file from Plink
  --population-file FILE
                        A population file containing the following columns
                        (without a header): FID, IID and POP. POP should be
                        one of 'CEU', 'JPT-CHB', 'YRI' and SOURCE.

Options:
  --outliers-of POP     Finds the outliers of this population. [default: CEU]
  --multiplier FLOAT    To find the outliers, we look for more than x times
                        the cluster standard deviation. [default: 1.9]
  --xaxis COMPONENT     The component to use for the X axis. [default: C1]
  --yaxis COMPONENT     The component to use for the Y axis. [default: C2]
  --format FORMAT       The output file format (png, ps, or pdf formats are
                        available). [default: png]

Output File:
  --out FILE            The prefix of the output files. [default: ethnicity]
```

5.12.6 The Algorithm

For more information about the actual algorithms and source codes, refer to the following pages.

- *pyGenClean.Ethnicity.check_ethnicity*
- *pyGenClean.Ethnicity.find_outliers*
- *pyGenClean.PlinkUtils.plot_MDS_standalone*

5.13 Minor Allele Frequency of Zero Module

The usage of the standalone module is shown below:

```
$ pyGenClean_flag_maf_zero --help
usage: pyGenClean_flag_maf_zero [-h] [-v] --bfile FILE [--out FILE]

Flags SNPs with MAF of 0.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --bfile FILE   The input file prefix (will find the plink binary files by
                 appending the prefix to the .bim, .bed and .fam files,
                 respectively.

Output File:
  --out FILE     The prefix of the output files. [default: flag_maf_0]
```

5.13.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.13.2 Procedure

Here are the steps performed by the module:

1. Computes the frequencies using Plink.
2. Finds markers with a MAF of zero.

5.13.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `flag_maf_0`]):

1. One file and one set of PLINK's result file:
   - `flag_maf_0.frq`: the frequency of each marker in the source dataset.
   - `flag_maf_0.list`: the list of markers with a minor allele frequency of zero.

5.13.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.FlagMAF.flag_maf_zero*

5.14 Hardy Weinberg Equilibrium Module

The usage of the standalone module is shown below:

```
$ pyGenClean_flag_hw --help
usage: pyGenClean_flag_hw [-h] [-v] --bfile FILE [--hwe FLOAT] [--out FILE]

Flags SNPs with Hardy-Weinberg disequilibrium.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --bfile FILE   The input file prefix (will find the plink binary files by
                 appending the prefix to the .bim, .bed and .fam files,
                 respectively.

Options:
  --hwe FLOAT    The Hardy-Weinberg equilibrium threshold. [default: 1e-4]

Output File:
  --out FILE     The prefix of the output files. [default: flag_hw]
```

5.14.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.14.2 Procedure

Here are the steps performed by the module:

1. Computes the number of markers in the input file.
2. Computes the Bonferroni threshold.
3. Runs Plink to find failed markers for HWE with the Bonferroni threshold.
4. Runs Plink to find failed markers for HWE with the default threshold.
5. Compares the two marker lists (Bonferroni and default threshold) and finds markers that are between the two thresholds.

5.14.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `flag_hw`]):

1. No files are created for this step.
2. No files are created for this step.
3. One set of PLINK's binary file is created:
   - `flag_hw.threshold_X.Xe-X`: the data set containing only the markers that pass the HWE test (above the Bonferroni threshold).
4. One set of PLINK's binary file is created:
   - `flag_hw.threshold_1e-4`: the data set containing only the markers that pass the HWE test (above the genome wide significance threshold of 1×10^{-4}). This value can be modified at the command line.
5. Three files are created:
   - `flag_hw.snp_flag_threshold_X.Xe-X`: the list of markers that failed HWE test for the Bonferroni threshold.
   - `flag_hw.snp_flag_threshold_1e-4`: the list of markers that failed HWE test for the genome wide significance threshold of 1×10^{-4}. This value can be modified at the command line.
   - `flag_hw.snp_flag_threshold_between_1e-4-X.Xe-X`: the list of markers that failed HWE test at a threshold between the Bonferroni and the genome wide significance thresholds, so that you can exclude only the ones that have a lower p value than the Bonferroni threshold.

5.14.4 The Algorithm

For more information about the actual algorithms and source codes, refer to the following page.

- *pyGenClean.FlagHW.flag_hw*

5.15 Comparison with a Gold Standard Module

Explain the procedure here.

5.15.1 Output Files

Describe the output files here.

5.15.2 The algorithm

For more information about the actual algorithms and source codes, refer to the following pages.

- *pyGenClean.Misc.compare_gold_standard*

5.16 Plink Utils

This module provides useful functions and scripts for efficient interactions with PLINK's output files. For example, the majority of PLINK's output files are spaced delimited, and are formated in such a way that it is "beautiful" to the human eye, but is a bit harder to parse using a script compared to tabulated files. The *pyGenClean.PlinkUtils.createRowFromPlinkSpacedOutput()* function helps producing an array of all the fields for each line.

5.16.1 Comparing BIM files

Another example is the fact that when PLINK removes a certain amount of markers from the data file, it just gives the number of excluded markers, but not a list. The *pyGenClean.PlinkUtils.compare_bim* module creates a list of markers that were removed from the original dataset when compared with the new one. Here is the usage of the standalone script:

```
$ pyGenClean_compare_bim --help
usage: pyGenClean_compare_bim [-h] [-v] --before FILE --after FILE
                              [--out FILE]

Compares BIM file.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Input File:
  --before FILE  The name of the bim FILE before modification.
  --after FILE   The name of the bim FILE after modification.

Output File:
  --out FILE     The prefix of the output files. [default: snp_removed]
```

5.16.2 Subsetting a dataset

A useful standalone script is the *pyGenClean.PlinkUtils.subset_data* module. It helps in subsetting a dataset by keeping or removing a set of samples, and at the same time extracting or excluding a set of markers. The following standalone script is available for the user:

```
$ pyGenClean_subset_data --help
usage: pyGenClean_subset_data [-h] [-v] --ifile FILE [--is-bfile] [--is-tfile]
                              [--is-file] [--exclude FILE] [--extract FILE]
                              [--remove FILE] [--keep FILE] [--out FILE]

Subsets genotype data using Plink.

optional arguments:
  -h, --help      show this help message and exit
  -v, --version   show program's version number and exit

Input File:
  --ifile FILE    The input file prefix. The format will be specified by --is-
                  bfile, --is-tfile or --is-file, for bfile, tfile and file,
                  respectively.
  --is-bfile      The file specified by --ifile is a bfile
  --is-tfile      The file specified by --ifile is a tfile
  --is-file       The file specified by --ifile is a file

Options:
  --exclude FILE  A file containing SNPs to exclude from the data set.
  --extract FILE  A file containing SNPs to extract from the data set.
  --remove FILE   A file containing samples (FID and IID) to remove from the
                  data set.
  --keep FILE     A file containing samples (FID and IID) to keep from the
                  data set.

Output File:
  --out FILE      The prefix of the output files. [default: subset]
```

The standalone script works with the three most used PLINK's format: pedfile, transposed and binary pedfiles. The `--is-bfile`, `--is-tfile` and `--is-file` options tell the standalone script what is the format of the input file. The output file format will be the same as the input one.

5.16.3 The Algorithm

For more information about the actual algorithms and source codes, refer to the following pages.

- *pyGenClean.PlinkUtils*
- *pyGenClean.PlinkUtils.compare_bim*
- *pyGenClean.PlinkUtils.subset_data*

CHAPTER

SIX

CITING PYGENCLEAN

If you use `pyGenClean` in any published work, please cite the published scientific paper describing the tool.

> Lemieux Perreault LP, Provost S, Legault MA, Barhdadi A, Dubé MP: **pyGenClean: efficient tool for genetic data clean up before association testing.** *Bioinformatics* 2013, 29(13):1704-1705 [DOI:10.1093/bioinformatics/btt261]

CHAPTER

SEVEN

APPENDIX

7.1 Result Summary Table

This table summurized information available from output files produced by *pyGenClean* during the data clean up procedure. Numbers correspond to number of lines in output files see *Proposed Protocol* for details. Only removed SNPs and IDs are indicated in the column SNPs and IDs, flagged SNPs or IDs are present in the n column.

Table 7.1: Summary information of the data clean up procedure.

Description	n	SNPs	IDs
Total number of SNPs in file received	2,379,855		
Total number of samples	494		
Number of duplicate samples	0		
Number of individuals with no genotype (failed)	0		
Number of SNPs with no physical position (chromosome and physical position = 0)	7,239	-7,239	
Number of INDEL	43	-43	
Number of replicate controls	5		
Number of replicate samples	0		
Number of duplicate SNPs (by chromosome and physical position)	5,643		
Duplicated SNPs by chromosome and physical position with the same allele (merge)	5,417	-5,147	
Number of duplicated SNP with <98% concordance	22	-22	
Completely failed SNPs	1	-1	
All heterozogous SNPs	0		
Number of individuals removed because they have more than 10% missing genotypes	5		-5
Number of SNPs removed because they have more than 2% missing value	128,562	-128,562	
Number of individuals removed because they have more than 2% missing genotypes	7		-7
Number of individuals with gender problem	1		
Number of SNPs with plate bias test P value below threshold of 1×10^{-7}	19		
Number of SNPs used for IBS analysis	73,651		
Number of duplicates pairs or twin	1		
Number of related pairs (including twins)	2		
Number of SNPs used for MDS analysis	80,262		
Number of individuals with ethnicity other than Caucasian as detected by MDS analysis	20		
Number of gender problems	1		-1

Continued on next page

Table 7.1 – continued from previous page

Description	n	SNPs	IDs
Number of related pairs	2		-2
Number of caucasian outliers	20		-20
Number of controls	5		-5
Number of heterozygote haploid genotypes set to missing (after correction of gender problems)	277,206		
Number of SNPs with MAF=0	602,480	-602,480	
Number of SNPS with HWE test P Value below threshold of 1×10^{-4} and higher than Bonferroni threshold	603		
Number of SNPS with HWE test below Bonferroni threshold	162	-162	
Total number of SNPs	**1,635,931**		
Total number of samples	**454**		

PYTHON MODULE INDEX

INDEX