EpiGEN: an epistasis simulation pipeline

User Guide

David B. Blumenthal, Lorenzo Viola, Markus List, Jan Baumbach, Paolo Tieri, and Tim Kacprowski

Contents:

---

1 Getting Started with EpiGEN

1.1 Scope

EpiGEN is an easy-to-use epistasis simulation pipeline written in Python. It supports epistasis models of arbitrary size, which can be specified either extensionally or via parametrized risk models. Moreover, the user can specify the minor allele frequencies (MAFs) of both noise and disease SNPs, and provide a biased target distribution for the generated phenotypes to simulate observation bias.

1.2 Installation

EpiGen is freely available on GitHub. To install it on your machine, simply execute the following line in a terminal:

```
git clone https://github.com/baumbachlab/epigen
```

1.3 Usage

The user interface of EpiGEN consists of three scripts:

- `simulate_data.py`
- `generate_genotype_corpus.py`
- `merge_genotype_corpora.py`

The script `simulate_data.py` simulates epistasis data on top of a pre-computed genotype corpus. For each chromosome `<CHROM>` and each HAPMAP3 population code `<POP>`, EpiGEN contains a pre-computed corpus for 10000 individuals, which is identified by the prefix `corpora/<CHROM>_<POP>_`. For example, if you want to generate epistasis data with ID 0 for 7500 individuals and 10000 SNPs on top of the pre-computed corpus `corpora/1_ASW_`, where the parametrized epistasis model `models/param_model.xml` acts upon the SNPs with IDs 156, 3, and 1076 in the corpus, you can use `simulate_data.py` as follows:

```
python3 simulate_data.py --sim-ids 0 --corpus-id 1 --pop ASW --inds 7500 --snps␣
↪10000 --disease-snps 156 3 1076 --model models/param_model.xml
```

As you will notice when executing this command, a large fraction of the runtime of `simulate_data.py` is used for loading the corpora. If you want to simulate data for only a small number of individuals, it is therefore advisable to first compute your own, smaller corpora. You can also speed-up the script by unzipping the corpora before running it.

If you want to use custom corpora instead of the pre-computed ones, you can generate them via the script `generate_genotype_corpus.py`. For example, the corpus `corpora/1_ASW_` shipped with EpiGEN was generated as follows:

```
python3 generate_genotype_corpus.py --corpus-id 1 --pop ASW --inds 10000 --chroms␣
↪1 --compress
```

Finally, the script `merge_genotype_corpora.py` allows you to merge pre-computed corpora into a larger copus. For instance, the following command merges the pre-computed corpora `corpora/1_ASW_` and `corpora/2_ASW_` into a newly generated corpus `corpora/23_ASW_`:

```
python3 merge_genotype_corpora.py --corpus-ids 1 2 --pops ASW ASW --corpus-id 23 --
↪append SNPS
```

Detailed descriptions of how to use the scripts can be found in the HTML and PDF documentations contained in `docs/build/html` and `docs/build/latex`.

1.4 Implementing Custom Interaction Models

EpiGEN natively supports four parametrized interaction models: exponential, multiplicative, joint-dominant, and joint-recessive interaction. Further interaction models can easily be implemented by the user. Assume, for instance, that the user wants to implement xor-dominant interaction, i.e., a parametrized interaction model where there is an effect if and only if there is at least one minor allele at exactly one of the SNPs involved in the interaction. Then it suffices to insert the following five lines of code at line 242 of `utils/parametrized_model.py`:

```
elif model_type == "xor-dominant":
        if np.sum(gen_at_snp_set[poss]) == 1:
                return alpha
        else:
                return 1
```

For consistency, it is also recommendable to add the string `"xor-dominant"` to the error message on line 249 of `utils/parametrized_model.py`, as well to the list of acceptable interaction types on line 42 of the document type definition `models/ParametrizedModel.dtd`.

1.5 Requirements

EpiGEN has the following dependencies:

- Python 3.3 or higher.
- Numpy 1.17.3 or higher.
- Scipy 1.3.1 or higher.

- Matplotlib 3.1.1 or higher.

Moreover, due to its HAPGEN2 dependency, the script `generate_genotype_corpus.py` needs to be run on a Linux machine or on a machine running macOS 10.14 or lower. However, you can avoid running `generate_genotype_corpus.py` by using the pre-computed corpora and merging them, if necessary.

If you want to re-compile the documentation contained in the `docs` directory, you additionally need to install Sphinx, the extension recommonmark, and the package mock. If you have these packages installed, the HTML and PDF documentations can be re-compiled by executing `make html` and `make latexpdf` from the `docs` directory.

1.6 License

All of EpiGEN's Python sources are licensed under the GNU General Public License 3. However, this license does not cover the HAPGEN2 binaries, which are distributed with EpiGEN and are called by the script `generate_genotype_corpus.py`. HAPGEN2 is property of the University of Oxford and may only be freely used for academic research and in accordance with the license found at https://mathgen.stats.ox.ac.uk/genetics_software/hapgen/LICENCE. Copies of the GNU General Public License 3 and of the license for HAPGEN2 are distributed with EpiGEN.

1.7 Citing EpiGEN

If you use EpiGEN, please cite the following paper:

- D. B. Blumenthal, L. Viola, M. List, J. Baumbach, P. Tieri, T. Kacprowski. "EpiGEN: an epistasis simulation pipeline". Submitted.

1.8 Structure of the Repository

```
.
├── README.md                          // README
├── LICENSE                            // A copy of the GNU General Public License 3
├── requirements.txt                   // Lists dependencies
├── simulate_data.py                   // Script to simulate epistasis data
├── generate_genotype_corpus.py        // Script to generate genotype corpus
├── merge_genotype_corpora.py          // Script to merge genotype corpora
├── test_runtime.py                    // Script to test EpiGEN's runtime performance
├── docs                               // Contains Sphinx documentation
├── sim                                // Output directory for simulated data
├── corpora                            // Output directory for genotype corpora
├── temp                               // Contains auxiliary files
├── ext                                // Contains external libraries and data
│   ├── HAPGEN2                        // Contains HAPGEN2 binaries and license
│   └── HAPMAP3                        // Contains HAPMAP3 data
├── models                             // Contains epistasis models
│   ├── ParametrizedModel.dtd          // Doctype definition for parametrized models
│   ├── ext_model.ini                  // An example of an extensional model
│   ├── param_model.xml                // An example of a parametrized model
│   └── ...                            // Further models
└── utils                              // Contains the core of EpiGEN
    ├── __init__.py                    // __init__ file
    ├── data_simulator.py              // Implements simulation of epistasis data
    ├── genotype_corpus_generator.py   // Implements generation of genotype corpora
    ├── genotype_corpusmerger.py       // Implements merging of genotype corpora
    ├── parametrized_model.py.         // Implements parametrized models
    ├── extensional_model.py.          // Implements extensional models
    └── argparse_checks.py.            // Implements argparse checks
```

2 The script `simulate_data.py`

This script constitutes the main user interface of EpiGEN – run it, to simulate epistasis data.

This script has to be run on top of a pre-computed genotype corpus. For each population code `<POP>` and each chromosome `<CHROM>`, EpiGEN contains a pre-computed corpus for 10000 individuals. These corpora can be selected by running the script with the options `--corpus-id <CRHOM> --pop <POP>`. You can also use your own corpora – simply run the script `generate_genotype_corpus.py` before running this script.

Usage:

```
python3 simulate_data.py [required arguments] [optional arguments]
```

Required Arguments:

`--corpus-id CORPUS_ID`

Description: ID of selected genotype corpus.

Accepted Arguments: Non-negative integers.

Effect: Together with `--pop`, this option selects the genotype corpus with the prefix `./corpora/<CORPUS_ID>_<POP>`. If this corpus does not exist, the script raises an error. If necessary, run the script `./generate_genotype_corpus.py` to generate the desired corpus.

`--pop POP`

Description: HAPMAP3 population code of selected genotype corpus.

Accepted Arguments: ASW, CEU, CEU+TSI, CHD, GIH, JPT+CHB, LWK, MEX, MKK, TSI, and MIX (for merged corpora).

Effect: Together with `--corpus-id`, this option selects the genotype corpus with the prefix `./corpora/<CORPUS_ID>_<POP>`. If this corpus does not exist, the script raises an error. If necessary, run the script `./generate_genotype_corpus.py` to generate the desired corpus.

`--model MODEL`

Description: Path to epistasis model given as INI or XML file. INI files are used to provide extensionally defined models, XML files specify parametrized models.

Accepted Arguments: Strings ending in `.ini` or `.xml` that represent paths to existing model files.

Effect: Specifies the epistasis model used by the simulator.

Format of INI files for extensionally defined models: For each genotype of length `<size>`, parameters of a Normal distribution must be provided. INI files for quantitative phenotypes have to be of the following format::

```
[Model Type]
size = <size>
phenotype = quantitative
[Model Definition]
0,...,0 = <mu>,<stdev>
.
.
.
2,...,2 = <mu>,<stddev>
```

For each genotype of length `<size>`, parameters of a categorical distribution must be provided INI files for categorical phenotypes have to be of the following format::

```
[Model Type]
size = <size>
phenotype = <c>
[Model Definition]
0,...,0 = <p_1>,...,<p_c>
.
.
.
2,...,2 = <p_1>,...,<p_c>
```

Format of XML files for parametrized models: XML files have to match the document type definition `models/ParametrizedModel.dtd`.

Examples: Cf. the files `models/ext_model.ini` and `models/param_model.xml`.

`--snps SNPS`:

Description: Number of SNPs in simulated data.

Accepted Arguments: Positive integers. If larger than the number of SNPs in the selected corpus, it is lowered to this number. Should be set to a number that is significantly smaller than the number of SNPs in the selected corpus, because otherwise, EpiGEN's subsampling techniques have no effect.

Effect: Determines how many SNPs from the selected corpus are included in the simulated data.

`--inds INDS`

Description: Number of individuals in simulated data.

Accepted Arguments: Positive integers. If larger than the number of individuals in the selected corpus, it is lowered to this number. Should be set to a number that is significantly smaller than the number of individuals in the selected corpus, because otherwise, EpiGEN's subsampling techniques have no effect.

Effect: Determines how many individuals from the selected corpus are included in the simulated data.

Required Group of Mutually Exclusive Arguments:

`--sim-ids SIM_ID [SIM_ID ...]`

Description: IDs of simulated data.

Accepted Arguments: One or several non-negative integers.

Effect: If N IDs <SIM_ID_1> ... <SIM_ID_N> are provided, N simulated epistasis instances with prefices `./sim/<SIM_ID_1>_<CORPUS_ID>_<POP>`, ..., `./sim/<SIM_ID_N>_<CORPUS_ID>_<POP>` are generated.

`--num-sims NUM_SIMS`

Description: The number of epistasis instances that should be generated.

Accepted Arguments: Positive integer.

Effect: If specified, <NUM_SIMS> simulated epistasis instances with prefices `./sim/0_<CORPUS_ID>_<POP>`, ..., `./sim/<NUM_SIMS-1>_<CORPUS_ID>_<POP>` are generated.

Optional Arguments:

`--global-maf-range LB UB`

Description: Range of acceptable MAFs for noise SNPs.

Accepted Arguments: Floats `<LB>` and `<UB>` with `0 <= <LB> < <UB> <= 1`.

Default: [0,1]

Effect: All SNPs except for the disease SNPs are randomly sampled from those SNPs in the corpus whose MAFs fall into the specified range. If the range is too narrow, it is dynamically extended at runtime.

`--biased-distr PARAM [PARAM ...]`

Description: Biased target distribution for simulated phenotypes.

Accepted Arguments for Quantitative Phenotypes: A white-space separated list of floats of length 2 whose elements represent the mean (first element) and standard deviation (second element) of a Normal distribution.

Accepted Arguments for Categorical Phenotypes: A white-space separated list of floats of length `<c>` whose entries represent the probabilities of the `<c>` categories.

Effect: If provided, the individuals are subsampled after generating the phenotypes such that the obtained phenotype distribution matches the biased distribution. This option can hence be used to model observation bias.

`--seed SEED`

Description: Seed for numpy.random.

Accepted Arguments: Non-negative integers.

Effect: If provided, the simulator always generates the same data given the same input.

`--compress`

Description: Compress the generated output files.

Accepted Arguments: None.

Effect: Determines the suffix `<SUFFIX>` of the generated files. If provided, `<SUFFIX>` is set to `json.bz2`. Otherwise, it is set to `json`.

`-h, --help`

Effect: Show help message and exit.

Optional Mutually Exclusive Arguments:

`--disease-snps SNP [SNP ...]`

Description: Position of disease SNPs in selected genotype corpus.

Accepted Arguments: White space separated list of non-negative integers whose length matches the size of the model specified via the option `--model`. All integers must be smaller than the number of SNPs in the selected corpus.

Effect: If provided, the selected SNPs form the disease SNP set employed by the simulator.

`--disease-maf-range LB UB`

Description: Range of acceptable MAFs for disease SNPs.

Accepted Arguments: Floats `<LB>` and `<UB>` with `0 <= <LB> < <UB> <= 1`.

Default: [0,1]

Effect: Unless `--disease-snps` is provided, the disease SNPs are randomly sampled from those SNPs in the corpus whose MAFs fall into the specified range. If the range is too narrow, it is dynamically extended at runtime.

Output:

Output File:

Path: `./sim/<ID>_<CORPUS_ID>_<POP>.<SUFFIX>`

Format for Quantitative Phenotypes: (Compressed) JSON file of the form `{"num_snps": <NUM_SNPS>, "num_inds": <NUM_INDS>, "model_type": "quantitative", "genotype": <GENOTYPE_DATA>, "phenotype": <PHENOTYPE_DATA>, "snps": <SNP_DATA>, "disease_snps": <DISEASE_SNP_DATA>, "mafs": <MAF_DATA>}`.

Format for Categorical Phenotypes: (Compressed) JSON file of the form `{"num_snps": <NUM_SNPS>, "num_inds": <NUM_INDS>, "model_type": "categorical", "num_categories": <NUM_CATEGORIES>, "genotype": <GENOTYPE_DATA>, "phenotype": <PHENOTYPE_DATA>, "snps": <SNP_DATA>, "disease_snps": <DISEASE_SNP_DATA>, "mafs": <MAF_DATA>}`.

<NUM_SNPS>

Key: `"num_spns"`

Content and Format: Integer representing the number of SNPs.

<NUM_INDS>

Key: `"num_inds"`

Content and Format: Integer representing the number of individuals.

<NUM_CATEGORIES>

Key: `"num_categories"`

Content and Format: Integer representing the number of categories for categorical phenotypes.

<GENOTYPE_DATA>

Key: `"genotype"`

Content and Format: JSON field of the form `[[G_0_0, ..., G_0_<INDS-1>] ... [G_<SNPS-1>_0, ..., G_<SNPS-1>_<INDS-1>]]`, where `G_S_I` encodes the number of minor alleles of the individual with index `I` at the SNP with index `S`.

<PHENOTYPE_DATA>

Key: `"phenotype"`

Content and Format: JSON field of the form `[P_0, ..., P_<INDS-1>]`, where `P_I` encodes the phenotype of the individual with index `I`.

<SNP_DATA>

Key: `"snps"`

Content and Format: JSON field of the form `[INFO_0, ..., INFO_<SNPS-1>]`, where `INFO_S` contains the following information about the SNP with index `S`: RS identifier, chromosome number, position on chromosome, major allele, minor allele.

<DISEASE_SNP_DATA>

Key: `"disease_snps"`

Content and Format: JSON field of the form `[S_0, ..., S_<MODELSIZE-1>]`, where `S_POS` encodes the index of the SNP at position `POS` in the epistasis model.

<MAF_DATA>

Key: `"mafs"`

Content and Format: JSON field of the form `[MAF_0, ..., MAF_<SNPS-1>]`, where `MAF_S` encodes the MAF of the SNP with index `S`.

simulate_data.**run_script**()

Runs the script.

3 The script `generate_genotype_corpus.py`

Run this script to generate genotype corpora.

For each HAPMAP3 population and each chromosome, EpiGEN comes with a pre-computed corpus for 10000 individuals. The corpora for chromosome <CHROM> have corpus ID <CHROM> and can be used by `simulate_data.py` right away. If you want to simulate data on top of your own corpora, run this script.

Usage:

```
python3 generate_genotype_corpus.py [required arguments] [optional arguments]
```

Requried Arguments:

`--corpus-id CORPUS_ID`

Description: ID of generated genotype corpus.

Accepted Arguments: Non-negative integers. If contained in range(1,23), the pre-computed corpora shipped with EpiGEN are overwritten.

Effect: Together with `--pop`, this option determines the prefix `./corpora/<CORPUS_ID>_<POP>` of the files that contain the generated corpus.

`--pop POP`

Description: HAPMAP3 population code of generated genotype corpus.

Accepted Arguments: ASW, CEU, CEU+TSI, CHD, GIH, JPT+CHB, LWK, MEX, MKK, and TSI.

Effect: Together with `--pop`, this option determines the prefix `./corpora/<CORPUS_ID>_<POP>` of the files that contain the generated corpus.

`--inds INDS`

Description: Number of individuals in the corpus.

Accepted Arguments: Positive integers.

Effect: Determines how many individuals are contained in the generated corpus.

Optional Arguments:

`--chroms CHROM [CHROM ...]`

Description: List of chromosomes for which the corpus should be generated.

Accepted Arguments: A white-space separated list of integers between 1 and 22.

Default: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22]

Effect: For each selected chromosome, a corpus is generated by calling HAPGEN2 without disease SNPs. The final corpus is then obtained by concatenating the corpora.

`--compress`

Description: Compress the generated output files.

Accepted Arguments: None.

Effect: Determines the suffix `<SUFFIX>` of the generated files. If provided, `<SUFFIX>` is set to `json.bz2`. Otherwise, it is set to `json`.

`-h, --help`

Effect: Show help message and exit.

Output:

Genotype Data:

File: `./corpora/<CORPUS_ID>_<POP>_genotype.<SUFFIX>`

Content and Format: Compressed JSON file of the form `[[G_0_0, ..., G_0_<INDS-1>], ..., [G_<SNPS-1>_0, ..., G_<SNPS-1>_<INDS-1>]]`, where `G_S_I` encodes the number of minor alleles of the individual with index `I` at the SNP with index `S`.

SNPs:

File: `./corpora/<CORPUS_ID>_<POP>_snps.<SUFFIX>`

Content and Format: Compressed JSON file of the form `[INFO_0, ..., INFO_<SNPS-1>]`, where `INFO_S` contains the following information about the SNP with index `S`: RS identifier, chromosome number, position on chromosome, major allele, minor allele.

MAFs:

File: `./corpora/<CORPUS_ID>_<POP>_mafs.<SUFFIX>`

Content and Format: Compressed JSON file of the form `[MAF_0, ..., MAF_<SNPS-1>]`, where `MAF_S` encodes the MAF of the SNP with index `S`.

Cumulative MAF distribution:

File: `./corpora/<CORPUS_ID>_<POP>_cum_mafs.<SUFFIX>`

Content and Format: Compressed ordered JSON file of the form `[[MAF_0, COUNT_0], ..., [MAF_<NMAFS-1>, COUNT_<NMAFS-1>]]`, where `COUNT_POS` encodes the number of SNPs is does not exceed `MAF_POS`.

Plot of Cumulative MAF distribution:

File: `./corpora/<CORPUS_ID>_<POP>_cum_mafs.pdf`

Content and Format: Plot of cumulative MAF distribution. Can be used to determine feasible ranges of MAFs passed to the options `--global-maf-range` and `--disease-maf-range` of the script `simulate_data.py`.

`generate_genotype_corpus.`**`run_script`**`()`
Runs the script.

4 The script `merge_genotype_corpora.py`

Run this script to merge pre-computed genotype corpora.

For each HAPMAP3 population and each chromosome, EpiGEN comes with a pre-computed corpus for 10000 individuals. The corpora for chromosome <CHROM> have corpus ID <CHROM> and can be used by `simulate_data.py` right away. If you want to these or other corpora, run this script.

Usage:

```
python3 merge_genotype_corpora.py [required arguments] [optional arguments]
```

Requried Arguments:

`--corpus-id CORPUS_ID`

Description: ID of merged genotype corpus.

Accepted Arguments: Non-negative integers. If contained in range(1,23), the pre-computed corpora shipped with EpiGEN are overwritten.

Effect: Together with `--pop`, this option determines the prefix `./corpora/<CORPUS_ID>_<POP>` of the files that contain the generated corpus.

`--pops POP POP [POP ...]`

Description: List of HAPMAP3 population codes of the corpora that should be merged. The size must match the size of the argument passed to `--corpus-ids`.

Accepted Arguments: ASW, CEU, CEU+TSI, CHD, GIH, JPT+CHB, LWK, MEX, MKK, TSI, and MIX.

Effect: Selects the corpora that should be merged and determines the prefix `./corpora/<CORPUS_ID>_<POP>` of the files that contain the generated corpus. If the list passed to this argument contains only one population code, `<POP>` is the set to this code. Otherwise, `<POP>` is set to `MIX`.

`--corpus-ids CORPUS_ID CORPUS_ID [CORPUS_ID ...]`

Description: The IDs of the corpora that should be merged.

Accepted Arguments: Lists of non-negative integers of size at least 2. The size must match the size of the argument passed to `--pops`.

Effect: Selects the corpora that should be merged.

`--append APPEND`

Description: The axis along which the corpora should be merged.

Accepted Arguments: "SNPS" or "INDS".

Effect: Set to "SNPS" to append the SNPs and to "INDS" to append the individuals.

Optional Arguments:

`--compress`

Description: Compress the generated output files.

Accepted Arguments: None.

Effect: Determines the suffix `<SUFFIX>` of the generated files. If provided, `<SUFFIX>` is set to `json.bz2`. Otherwise, it is set to `json`.

`-h, --help`

Effect: Show help message and exit.

Output:

Genotype Data:

File: `./corpora/<CORPUS_ID>_<POP>_genotype.<SUFFIX>`

Content and Format: (Compressed) JSON file of the form `[[G_0_0, ..., G_0_<INDS-1>], ..., [G_<SNPS-1>_0, ..., G_<SNPS-1>_<INDS-1>]]`, where `G_S_I` encodes the number of minor alleles of the individual with index `I` at the SNP with index `S`.

SNPs:

File: `./corpora/<CORPUS_ID>_<POP>_snps.<SUFFIX>`

Content and Format: (Compressed) JSON file of the form `[INFO_0, ..., INFO_<SNPS-1>]`, where `INFO_S` contains the following information about the SNP with index `S`: RS identifier, chromosome number, position on chromosome, major allele, minor allele.

MAFs:

File: `./corpora/<CORPUS_ID>_<POP>_mafs.<SUFFIX>`

Content and Format: (Compressed) JSON file of the form `[MAF_0, ..., MAF_<SNPS-1>]`, where `MAF_S` encodes the MAF of the SNP with index `S`.

Cumulative MAF distribution:

File: `./corpora/<CORPUS_ID>_<POP>_cum_mafs.<SUFFIX>`

Content and Format: (Compressed) ordered JSON file of the form `[[MAF_0, COUNT_0], ..., [MAF_<NMAFS-1>, COUNT_<NMAFS-1>]]`, where `COUNT_POS` encodes the number of SNPs is does not exceed `MAF_POS`.

Plot of Cumulative MAF distribution:

File: `./corpora/<CORPUS_ID>_<POP>_cum_mafs.pdf`

Content and Format: Plot of cumulative MAF distribution. Can be used to determine feasible ranges of MAFs passed to the options `--global-maf-range` and `--disease-maf-range` of the script `simulate_data.py`.

`merge_genotype_corpora.`**`run_script`**`()`
Runs the script.

5 The script `test_runtime.py`

Run this script to test to carry out runtime tests.

Usage:

```
python3 test_runtime.py
```

Output:

File: `./num_inds_vs_runtime.csv`

Content: Runtimes for simulating 1 dataset with varying number of individuals with number of SNPs fixed to 100000.

File: `./num_snps_vs_runtime.csv`

Content: Runtimes for simulating 1 dataset with varying number of SNPs with number of individuals fixed to 10000.

File: `./num_datasets_vs_runtime.csv`

Content: Runtimes for simulating varying number of datasets with number of SNPs and individuals fixed to 100.

`test_runtime.`**`run_script`**`()`
Runs the script.

6 The package `utils`

This package contains the core of EpiGEN. Do not modify it, unless you really know what you are doing.

6.1 The module `utils.data_simulator.py`

Contains definition of DataSimulator class.

`class` `utils.data_simulator.`**`DataSimulator`**(*corpus_id*, *pop*, *model*, *num_snps*, *num_inds*, *disease_snps*, *biased_distr*, *noise_maf_range*, *disease_maf_range*, *seed*, *compress*)

Bases: `object`

Simulates epistasis data given a pre-generated genotype corpus.

This class is employed by the script simulate_data.py. Not intended for use outside of this script. Expects to be imported from EpiGEN's root directory.

`genotype`
A numpy.array with entries from range(3) whose rows represent SNPs and whose columns represent individuals contained in simulated data.

Type numpy.array

snps
A list with one entry for each row of self.genotype. The entries provide information about the corresponding SNP.

Type list of (list of str)

mafs
A numpy.array of floats representing the MAFs of all rows of self.genotype.

Type numpy.array

cum_mafs
A list of pairs of the form (MAF,count) representing the cumulative MAF distribution of self.mafs.

Type list of (float,int)

corpus_genotype
A numpy.array with entries from range(3) whose rows represent SNPs and whose columns represent individuals contained in the genotype corpus.

Type numpy.array

corpus_snps
A list with one entry for each row of self.corpus_genotype. The entries provide information about the corresponding SNP.

Type list of (list of str)

corpus_mafs
A numpy.array of floats representing the MAFs of all rows of self.corpus_genotype.

Type numpy.array

corpus_cum_mafs
A list of pairs of the form (MAF,count) representing the cumulative MAF distribution of self.corpus_mafs.

Type list of (float,int)

model
The epistasis model.

Type ExtensionalModel/ParametrizedModel

sim_id
An integer that represents the ID of the generated data.

Type int

corpus_id
An integer that represents the ID of the genotype corpus on top of which the data should be simulated.

Type int

pop
A string representing the HAPMAP3 population for which the selected genotype corpus was generated.

Type str

num_snps
The number of SNPs in the simulated data.

Type int

num_inds
The number of individuals in the simulated data.

Type int

total_num_snps
The number of SNPs in the selected genotype corpus.

Type int

total_num_inds
The number of individuals in the selected genotype corpus.

Type int

biased_distr
Parameters of biased observed phenotype distribution. If empty, no observation bias is applied.

Type list of float

phenotype
A numpy.array that stores the generated phenotypes.

Type numpy.array

disease_snps
A list of positions of the selected disease SNPs in self.snps and self.genotype.

Type list of int

input_disease_snps
A user-specified list of positions of the selected disease SNPs in self.snps and self.genotype.

Type list of int

noise_maf_range
A tuple of floats between 0 and 1 that specifies the range of acceptable MAFs for the selected noise SNPs.

Type float,float

disease_maf_range
A tuple of floats between 0 and 1 that specifies the range of acceptable MAFs for the selected disease SNPs.

Type float,float

epsilon
A small positive real used for comparing floats.

Type float

compress
If True, the simulated data is compressed.

Type bool

__init__(*corpus_id*, *pop*, *model*, *num_snps*, *num_inds*, *disease_snps*, *biased_distr*, *noise_maf_range*, *disease_maf_range*, *seed*, *compress*)
Initialized DataSimulator.

Parameters

- **corpus_id** (*int*) – An integer that represents the ID of the genotype corpus on top of which the data should be simulated.
- **pop** (*str*) – A string representing the HAPMAP3 population for which the selected genotype corpus was generated.
- **model** (*str*) – Path to an INI or an XML file containing the epistasis model specification.
- **num_snps** (*int*) – The number of SNPs in the simulated data.
- **num_inds** (*int*) – The number of individuals in the simulated data.

- **disease_snps** (*list of int*) – A list the disease SNPs' position in the selected genotype corpus. If not empty, its size must match the size of the model.
- **biased_distr** (*list of float*) – Parameters of biased observed phenotype distribution. If empty, no observation bias is applied.
- **noise_maf_range** (*float, float*) – A tuple of floats between 0 and 1 that specifies the range of acceptable MAFs for the selected noise SNPs.
- **disease_maf_range** (*float, float*) – A tuple of floats between 0 and 1 that specifies the range of acceptable MAFs for the selected disease SNPs.
- **seed** (*int/None*) – The seed for numpy.random (possibly None).
- **compress** (*bool*) – If True, the simulated data is compressed.

dump_simulated_data()
Dumps the simulated data.

generate_phenotype()
Generates the phenotype and adjusts the number of individuals.

sample_snps()
Samples the SNPs and the disease SNP set based on the MAFs.

set_sim_id(*sim_id*)
Sets the simulation ID and prepares next simulation on top of pre-loaded corpus.

Parameters **sim_id** (*int*) – An integer that represents the ID of the generated data.

6.2 The module utils.genotype_corpus_generator.py

Contains definition of GenotypeCorpusGenerator class.

class utils.genotype_corpus_generator.**GenotypeCorpusGenerator**(*chroms*, *num_inds*, *corpus_id*, *pop*, *compress*)

Bases: object

Generates genotype corpus by calling HAPGEN2 and merging the obtained chromosome-wise genotypes.

This class is employed by the script generate_genotype_corpus.py. Not intended for use outside of this script. Expects to be imported from EpiGEN's root directory.

hapmap_dir
The path to the HAPMAP3 directory.

Type str

hapgen_dir
The path to the directory that contains the HAPGEN2 binary.

Type str

chroms
A list of integers between 1 and 22 representing the chromosomes for which HAPGEN2 generates the genotypes.

Type list of int

num_inds
An integer that represents the number of individuals for which genotypes are constructed.

Type int

num_snps
An integer that represents the number of SNPs in the generated corpus.

Type int

corpus_id
An integer that represents the ID of the generated corpus.

Type int

pop
A string representing the HAPMAP3 population.

Type str

dummy_disease_snps
A dict that, for each chromosome, specifies the position of a SNP that can be passed to HAPGEN2 as argument of the -dl option.

Type dict of (int,int)

genotype
A numpy.array with entries from range(3) that contains the merged genotypes. The rows represent SNPs, the columns represent individuals.

Type numpy.array

snps
A list with one entry for each row of self.genotype. The entries provide information about the corresponding SNP.

Type list of (list of str)

mafs
A numpy.array of floats representing the MAFs of all rows of self.genotype.

Type numpy.array

cum_mafs
A list of pairs of the form (MAF,count) representing the cumulative MAF distribution.

Type list of (float,int)

compress
If True, the generated corpus is compressed.

Type bool

__init__(*chroms*, *num_inds*, *corpus_id*, *pop*, *compress*)
Initializes GenotypeCorpusGenerator.

Parameters

- **chroms** (*list of int*) – A list of integers between 1 and 22 representing the chroms for which HAPGEN2 should be called. Duplicates are ignored.
- **num_inds** (*int*) – An integer representing the number of individuals for which genotypes should be constructed.
- **corpus_id** (*int*) – An integer that represents the ID of the generated corpus.
- **compress** (*bool*) – If True, the generated corpus is compressed.

call_hapgen2()
Calls HAPGEN2 to generate genotypes for all chromosomes.

compute_mafs()
Computes MAFs for all SNPs contained in self.genotype.

dump_corpus()
Dumps the generated corpus to zipped JSON files.

merge_hapgen2_output()
Merges the output of HAPGEN2.

6.3 The module utils.genotype_corpus_merger.py

Contains definition of GenotypeCorpusMerger class.

class utils.genotype_corpus_merger.**GenotypeCorpusMerger**(*corpus_ids*, *pops*, *corpus_id*, *axis*, *compress*)

Bases: object

Merges pre-computed genotype corpora.

This class is employed by the script generate_genotype_corpus.py. Not intended for use outside of this script. Expects to be imported from EpiGEN's root directory.

corpus_ids
A list of integers that represents the IDs of the corpora that should be merged.

Type list of int

pops
A list of strings representing the HAPMAP3 population codes of the corpora that should be merged.

Type list of str

corpus_id
An integer that represents the ID of the generated corpus.

Type int

pop
A string representing the HAPMAP3 population code of the merged corpus.

Type str

genotype
A numpy.array with entries from range(3) that contains the merged genotypes. The rows represent SNPs, the columns represent individuals.

Type numpy.array

snps
A list with one entry for each row of self.genotype. The entries provide information about the corresponding SNP.

Type list of (list of str)

mafs
A numpy.array of floats representing the MAFs of all rows of self.genotype.

Type numpy.array

cum_mafs
A list of pairs of the form (MAF,count) representing the cumulative MAF distribution.

Type list of (float,int)

axis
An integer representing the axis of the merge (0 for merge along SNPs, 1 for merge along individuals).

Type int

num_snps
Number of SNPs in merged corpus.

Type int

num_inds
Number of individuals in merged corpus.

Type int

compress
If True, the merged corpus is compressed.

Type bool

__init__(*corpus_ids*, *pops*, *corpus_id*, *axis*, *compress*)
Initializes GenotypeCorpusGenerator.

Parameters

- **corpus_ids** (*list of int*) – A list of integers that represents the IDs of the corpora that should be merged.
- **pop** (*str*) – A list of strings representing the HAPMAP3 population codes of the corpora that should be merged.
- **corpus_id** (*int*) – An integer that represents the ID of the generated corpus.
- **axis** (*int*) – An integer representing the axis of the merge (0 for merge along SNPs, 1 for merge along individuals)
- **compress** (*bool*) – If True, the merged corpus is compressed.

compute_mafs()
Computes MAFs for all SNPs contained in self.genotype.

dump_corpus()
Dumps the generated corpus to zipped JSON files.

merge_corpora()

6.4 The module utils.extensional_model.py

Contains definition of ExtensionalModel class.

class utils.extensional_model.**ExtensionalModel**(*model*, *seed*)
Bases: object

Represents an extensionally defined epistasis model.

Extensional models can be used to model binary or non-binary categorical phenotypes, as well as quantitative phenotypes. This class is employed by the class DataSimulator. Not intended for use outside of this class.

size
An integer greater equal 2 representing the size of the model.

Type int

model
A numpy.array of dimension self.size representing the epistasis model.

Type numpy.array

phenotype
An integer greater equal 2 (for categorical phenotypes) or the string "quantitative" (for quantitative phenotypes).

Type int/str

__init__(*model*, *seed*)
Initializes the ExtensionalModel.

Parameters

- **model** (*str*) – Path to an INI file that contains the full extensional definiton of an epistasis model. For categorical models, a discrete probability distribution must be provided for each possible genotype of dimension self.size. For quantitative models, the mean and the standard deviation of normal distributions must be provided. Examples can be found in the directory epigen/models/.
- **seed** (*int/None*) – The seed for np.random (possibly None).

6.5 The module utils.parametrized_model.py

Contains definition of ParametrizedModel class.

class utils.parametrized_model.**ParametrizedModel**(*model*, *seed*)
Bases: object

Represents a parametrized epistasis model.

Parametrized models can be used to model binary categorical phenotypes. This class is employed by the class DataSimulator. Not intended for use outside of this class.

size
An integer greater equal 2 representing the size of the model.

Type int

baseline_alpha
A float greater 0 representing the baseline risk.

Type float

marginal_models
The marginal models.

Type dict of (int,callable)

interaction_models
The interaction models.

Type dict of ((list of int),callable)

phenotype
The inetger 2 (for dichotomous phenotypes) or the string "quantitative" (for quantitative phenotypes).

Type int/str

__init__(*model*, *seed*)
Initializes the ExtensionalModel.

Parameters

- **model** (*str*) – Path to an XML file that contains the specification of the parametrized model. Examples can be found in the directory epigen/models/.
- **seed** (*int/None*) – The seed for np.random (possibly None).

6.6 The module utils.argparse_checks.py

utils.argparse_checks.**check_interval**(*argname*)
Ensures that the provided arguments specify a sub-interval of [0,1].

Parameters **argname** (*str*) – Name of the argparse argument.

utils.argparse_checks.**check_length**(*argname*)
Ensures that at least two arguments are provided.

Parameters **argname** (*str*) – Name of the argparse argument.

utils.argparse_checks.**check_non_negative**(*argname*)
Ensures that the provided argument is non-negative.

Parameters **argname** (*str*) – Name of the argparse argument.

utils.argparse_checks.**check_positive**(*argname*)
Ensures that the provided argument is positive.

Parameters **argname** (*str*) – Name of the argparse argument.

Python Module Index

Index

Symbols

A

B

C

D

E

G

H

I

M

N

P

R

S

T

U