**Programming process steps in aver\_per\_sim\_1\_haplo\_min.py**

Royal Truman 22.10.2025

Contents

[Files linked to *aver\_per\_sim\_1\_haplo.py* 1](#_Toc212062128)

[Overview of processing steps 3](#_Toc212062129)

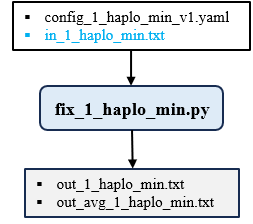
[Content of file haplo\_AVE\_A\_per\_gen\_1H.txt 5](#_Toc212062130)

[Content of file haplo\_AVE\_a\_\_per\_gen\_1H.txt 6](#_Toc212062131)

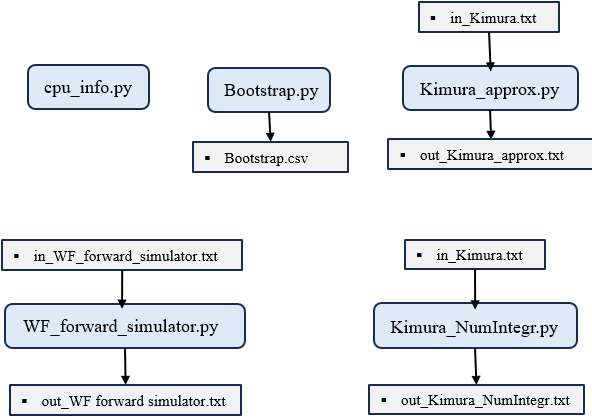
[Content of file A\_haplo\_AVE\_vals\_per\_gen\_1H.txt 7](#_Toc212062132)

# Files linked to *aver\_per\_sim\_1\_haplo\_min.py*

Figure 1 shows the file dependencies of python program *aver\_per\_sim\_1\_haplo\_min.py* and figure 2 the dependencies of various tools associated with this program.



**Figure 1**. Input and output file dependencies for python program *aver\_per\_sim\_1\_haplo.py*.



**Figure 2**. Input and output file dependencies for additional tools.

# Overview of processing steps

Here is a step-by-step chronological documentation of what the Python program *aver\_per\_sim\_1\_haplo.py* does:

(1) Import required modules: `numpy`, `os`, `sys`, `time`, `multiprocessing`, and `yaml`.

(2) Load configuration settings from the file `config\_1\_haplo\_min\_v1.yaml`, extracting `Repetitions` (number of repeats per simulation) and `max\_generations` (maximum generations per attempt).

(3) Define constants: a master random seed (`MASTER\_SEED = 42`), input filename (`in\_1\_haplo\_min.txt`), and output filenames for individual and averaged results.

(4) On Windows, prevent system sleep during execution by calling the Windows API via `ctypes`.

(5) Record the program start time to measure total execution duration.

(6) Define helper functions that return formatted column headings for the input file, individual results file, and averaged results file.

(7) Check whether the input file `in\_1\_haplo\_min.txt` exists; if not, create it with a header and example rows, print instructions, and exit.

(8) Read all non-empty lines from the input file, stripping leading/trailing whitespace.

(9) If the cleaned input contains no data lines, recreate the file with example rows, print instructions, and exit.

(10) Verify that the first line matches the expected header; if not, replace the file with a valid template, print instructions, and exit.

(11) If the file contains only the header (no parameter rows), append example rows, print instructions, and exit.

(12) Parse each subsequent line to extract four parameters: population size (`N`), selection coefficient (`s`), initial allele frequency (`p0`), and number of simulation attempts (`attempts`).

(13) Validate each parameter:

 – `N` must be an integer in [1, 10¹²],

 – `s` must be a float in [–2, 2],

 – `p0` must be a float in [0.0, 1.0],

 – `attempts` must be an integer in [1, 10¹²].

If any line fails validation, print an error message and mark the file as invalid.

(14) If any validation errors occurred, print a correction prompt and exit; otherwise, store all valid parameter tuples `(N, s, p0, attempts)` in `valid\_data`.

(15) Define the `simulate\_population` function, which:

 – Initializes counters for allele loss and fixation,

 – Sets a fixation threshold just below 1 (`p\_fix = 1 – 1/(2N)`),

 – For each simulation attempt:

  • Starts with allele frequency `p0`,

  • Iterates up to `max\_generations`:

   – Stops early if the allele is lost (`p\_t = 0`) or fixed (`p\_t > p\_fix`), recording the outcome and generation number,

   – Otherwise, computes Hardy–Weinberg genotype frequencies,

   – Applies viability selection using assigned fitness values,

   – Calculates the post-selection allele frequency,

   – Applies genetic drift via binomial sampling of `2N` alleles.

(16) Define the `worker` function, which:

 – Sets a deterministic random seed based on simulation index and repetition number (ensuring reproducibility across runs),

 – Calls `simulate\_population` with the assigned parameters,

 – Returns a tuple containing all input parameters and simulation outcomes.

(17) In the main execution block, determine the maximum number of parallel processes based on available CPU cores.

(18) Generate a list of simulation jobs, where each job corresponds to one repetition of one parameter set from `valid\_data`.

(19) Execute all jobs in parallel using a `multiprocessing.Pool`, collecting the results.

(20) Sort the results by simulation index and repetition number to ensure consistent output ordering.

(21) Aggregate raw results across all repetitions for each unique parameter set into a dictionary (`pooled\_data`), summing losses, fixations, and fixation generations, and preserving parameter metadata.

(22) Compute pooled statistics per parameter set:

 – Total trials = `Repetitions × attempts`,

 – Probabilities of loss and fixation = total events / total trials,

 – Standard error of fixation probability (using binomial variance),

 – Mean and sample standard deviation of generations to fixation (or `NaN` if no fixations occurred).

(23) Format individual repetition results into semicolon-separated strings with appropriate numerical precision.

(24) Attempt to delete any existing individual results file (`out\_1\_haplo\_min.txt`), then write a new file with a header and all formatted result lines; skip writing if file access fails.

(25) If `Repetitions > 1`, similarly attempt to write pooled average results to `out\_avg\_1\_haplo\_min.txt`, including aggregated statistics and the total number of repetitions; skip if file access fails.

(26) Compute total execution time and print it to the console.