**Sim generations population genetics toolkit**

Royal Truman

Download programs from: <https://github.com/RoyalTr/Sim_generations>

Contents

[Introduction 1](#_Toc211452667)

[Simulation Process 1](#_Toc211452668)

[Content of input file in\_2\_haplos.txt 1](#_Toc211452669)

[Output files generated using fix\_2\_haplos.py as an example 2](#_Toc211452670)

[Technical details 2](#_Toc211452671)

[Description of program steps 2](#_Toc211452672)

# Introduction

This collection of python program are built a round Wright-Fischer forward simulations of population genetics. Diploid organism are assumed along with Mendelian segregation of chromosomes under random mating throughout the population.

The most common genotype fitness model is assumed:

*wAA* = 1 + *s*, *wAa* = 1 + *hs*, *waa* = 1,

where *A* and *a* are alternative haplotypes at the same locus.

The usual goal of the simulations is to calculate the fixation probability of A/a and B/b alleles, and the number of generations required. The programs can simulate assumptions used in evolutionary and young earth creationist models, specifically regarding initial allele proportions and selection coefficients.

**Population Growth**: Population size is fixed (*r* = 0) or increases / decreases according to the \*\*discrete Beverton-Holt model\*\*.

**Fitness Calculation**: The average fitness of the population is recalculated every generation, based on the relative fitness values of the three genotypes (AA, Aa, and aa) and their current proportions within the population.

# Simulation Process

The programs can run independent simulations (controlled by variable Repetitions in config file) for various user-defined scenarios specified in simulation input files. Each scenario is configured using parameters such as the following for fix\_2\_haplos.py.

## Content of input file in\_2\_haplos.txt

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ni | r | K | s\_A | s\_B | p\_A\_i | p\_B\_i | h\_A | h\_B | attempts |
| 10 | 0.04 | 10,000 | **0** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |
| 10 | 0.04 | 10,000 | **0.02** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |
| 10 | 0.04 | 10,000 | **0.04** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |
| 10 | 0.04 | 10,000 | **0.06** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |
| 10 | 0.04 | 10,000 | **0.08** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |
| 10 | 0.04 | 10,000 | **0.1** | 0.001 | 0.01 | 0.01 | 0.5 | 0 | 1,000 |

*Ni*: Initial population size - r: Population growth rate

*r*: Growth rate

*K*: Carrying capacity, the maximum population inthat environment

*s\_A*: Selection coefficient for haplotype *A*

*s\_B*: Selection coefficient for haplotype *B*

*p\_A\_i*: Initial frequency of haplotype *A*

*p\_B\_i*: Initial frequency of haplotype *B*

*h\_A*: Dominance coefficient for haplotype *A*

*h\_B*: Dominance coefficient for haplotype *B*

Attempts: Number of simulation attempts to run x Repetitions

For each simulation attempt, the program iteratively calculates the allele frequencies, genotype frequencies, and mean population fitness. It then uses a binomial distribution to model random genetic drift and determine the allele frequencies in the next generation. The simulation continues until one of the alleles fixes (reaches a frequency of 0 or effectively 1).

## Output files generated using fix\_2\_haplos.py as an example

out\_2\_haplos.txt and out\_2\_haplos\_avg.txt contain key statistics from the simulations. Values stored include the probability of fixation for both alleles, the average number of generations to fixation, and the standard deviations for these metrics.

In addition, if the parameter document\_results\_every\_generation is set to true in config\_2\_haplos\_v1.yaml, then key values for each generation are stored in out\_2\_haplos\_per\_gen.txt. These provide the source input data for create\_plots\_ALL\_simul\_2\_haplos.py and aver\_per\_sim\_2\_haplos.py.

# Technical details

The program leverages multiprocessing to run simulations in parallel, optimizing execution time. The value of Repetitions in config\_2\_haplos\_v1.yaml specifies the maximum number of workers that will be running concurrently.

# Description of program steps

Tables 1 – 3 state the key processing steps of the three most important programs.

**Table 1**. Process steps in fix\_1\_haplo\_min.py

1. Import necessary Python modules such as numpy, os, sys, time, multiprocessing, and yaml.

2. Load configuration settings from the file "config\_1\_haplo\_min\_v1.yaml".

3. Extract the values for "Repetitions" and "max\_generations" from the configuration.

4. Define constants including MASTER\_SEED, input/output filenames, and example parameter rows.

5. Prevent system sleep on Windows using Windows API calls.

6. Record the program start time.

7. Define a function returning the expected input file header string.

8. Define a function returning the detailed results file header string.

9. If the input file does not exist, create it with headers and example rows, then exit.

10. Read all non-empty lines from the input file.

11. If the input file is empty, repopulate it with headers and example rows, then exit.

12. If the first line does not match the expected header, replace the file and exit.

13. If the input file contains only the header, append example rows and exit.

14. Validate that each data line has exactly four semicolon-separated fields.

15. Validate that population size (N) is a positive integer within the allowed range.

16. Validate that the selection coefficient (s) is a float between –2 and 2.

17. Validate that the initial allele frequency (p0) is a float between 0.0 and 1.0.

18. Validate that the number of attempts is a positive integer within the allowed range.

19. Store valid parameter sets in a list; exit if any errors are found.

20. Define the simulate\_population function to model allele dynamics under selection and drift.

21. Initialize counters for allele loss, fixation, and a list for fixation generations.

22. For each attempt, simulate up to max\_generations, updating allele frequency each generation.

23. Compute genotype frequencies assuming Hardy–Weinberg equilibrium.

24. Assign fitness values based on the selection coefficient s.

25. Calculate mean fitness and post-selection allele frequency.

26. Apply genetic drift via binomial sampling of allele counts.

27. Record loss if allele frequency hits 0; record fixation and generation if near fixation.

28. Define a worker function that sets a unique random seed and runs simulate\_population.

29. Determine the number of available CPU cores.

30. Generate a job list for all parameter-repetition combinations.

31. Execute jobs in parallel using multiprocessing and collect results.

32. Sort results by simulation number and repetition.

33. Aggregate raw results across repetitions for each parameter set using a defaultdict.

34. Sum losses, fixations, and fixation generations; store parameter metadata.

35. Compute pooled statistics: fixation/loss probabilities, standard error, and mean/std of fixation time.

36. Format individual-repetition result lines with per-rep statistics.

37. Delete any existing individual results file and write a new one with headers and data.

38. Skip writing the individual results file if access is denied and print a warning.

39. If Repetitions > 1, write pooled average results to "out\_avg\_1\_haplo\_min.txt".

40. Print total execution time.

**Table 2**. Process steps in fix\_1\_haplo.py

1. Import necessary Python modules including numpy, os, sys, time, multiprocessing, tempfile, and warnings.

2. Record the program start time.

3. Suppress runtime warnings about taking the mean of an empty slice.

4. Define a hardcoded MASTER\_SEED for reproducible results.

5. Specify filenames for configuration, input, and output files.

6. Attempt to import the yaml module; exit with an error if it is not installed.

7. Check if the configuration file exists; exit with an error if missing.

8. Load and parse the YAML configuration file; exit if parsing fails.

9. Extract and validate Repetitions, max\_generations, and document\_results\_every\_generation from the config; exit on missing or invalid values.

10. Prevent system sleep on Windows using Windows API calls.

11. Define the expected input file header string with seven parameters.

12. Define the detailed results file header string.

13. Define the averaged results file header string.

14. If the input file does not exist, create it with headers and example rows, then exit.

15. Read all non-empty lines from the input file.

16. If the input file is empty, repopulate it with headers and example rows, then exit.

17. Verify that the first line matches the expected header; exit if it does not.

18. If only the header is present, append example rows and exit.

19. Validate each data line to ensure it contains exactly seven semicolon-separated fields.

20. Validate that initial population size (Ni) is a positive integer within allowed bounds.

21. Validate that growth rate (r) is greater than –1.0.

22. Validate that carrying capacity (K) is an integer ≥ Ni.

23. Validate that selection coefficient (s\_A) is a float between –2 and 2.

24. Validate that the number of attempts is a positive integer within allowed bounds.

25. Validate that dominance coefficient (h\_A) is a float between –1 and 1.

26. Validate that initial allele frequency (p\_A\_i) is between 0.0 and 1.0.

27. Store valid parameter sets in a list; exit if any validation errors occurred.

28. Define the simulate\_population function to model allele dynamics with density-dependent population growth and selection.

29. Initialize counters and accumulators for fixation events and population sizes.

30. Precompute constants for population growth (r1 and rK).

31. If per-generation output is enabled, open a temporary file for writing per-generation data.

32. Precompute genotype fitness values based on s\_A and h\_A.

33. For each simulation attempt, initialize population size and allele frequency.

34. For each generation, compute and write current state (N, allele frequencies) to the per-generation file if enabled.

35. After writing, check for fixation (p\_A\_t = 0 or 1); record fixation type, generation, and population size if detected.

36. If no fixation, compute genotype frequencies, mean fitness, and post-selection allele frequency.

37. Update population size using Beverton-Holt (r ≥ 0) or exponential decay (r < 0), with stochastic rounding.

38. Break simulation if population size drops to zero or below.

39. Apply genetic drift via binomial sampling to determine next-generation allele frequency.

40. Include a defensive check to break if allele frequency becomes NaN.

41. Close the per-generation temporary file if it was opened.

42. Compute and return summary statistics: fixation probabilities, average fixation generations, and average final population sizes, along with the temp filename.

43. Define a worker function that sets a unique random seed per (SimNr, Rep) and calls simulate\_population.

44. In the main block, determine the number of available CPU cores.

45. If per-generation output is enabled, initialize the per-generation results file with a header.

46. Initialize the individual results file with a header and delete any existing version.

47. If Repetitions > 1, initialize the averaged results file with a header and delete any existing version.

48. Loop over each valid parameter set (SimNr).

49. For each SimNr, create a job for every repetition.

50. Execute all repetitions for the current SimNr in parallel using multiprocessing, preserving order.

51. If per-generation output is enabled, read each repetition’s temporary file in order and append its contents to the main per-generation results file.

52. Delete each temporary per-generation file after successful reading.

53. Write individual repetition results (fixation stats) to the main results file in repetition order.

54. Compute a combined homozygous/heterozygous loss metric as a weighted average of fixation outcomes.

55. If Repetitions > 1, aggregate raw counts across all repetitions to compute unbiased overall averages.

56. Reconstruct total fixation counts and sums from per-repetition results to avoid averaging bias.

57. Calculate true overall fixation probabilities, mean fixation generations, and mean final population sizes.

58. Compute the combined loss metrics using the overall averages, handling edge cases where one allele fixes with probability 1.

59. Write the averaged results for the current SimNr to the averaged results file.

60. Print confirmation messages indicating where output files were saved.

61. Print the total program execution time.

**Table 3**. Process steps in fix\_2\_haplos.py

1. Import necessary Python modules including numpy, os, sys, time, multiprocessing, shutil, and warnings.

2. Record the program start time.

3. Suppress runtime warnings about taking the mean of an empty slice.

4. Define a hardcoded MASTER\_SEED for reproducible simulation results.

5. Specify the configuration filename as "config\_2\_haplos\_v1.yaml".

6. Attempt to import the yaml module; exit with an error message if it is not installed.

7. Check if the configuration file exists; exit if it is missing.

8. Load and parse the YAML configuration file; exit if parsing fails.

9. Extract and validate Repetitions, max\_generations, and document\_results\_every\_generation from the config; exit on missing or invalid values.

10. Prevent system sleep on Windows using Windows API calls.

11. Define the expected input file header string with ten parameters for two loci.

12. Define the detailed results file header string with statistics for both loci and pan-homozygous outcomes.

13. Define the averaged results file header string matching the detailed results format.

14. Define the per-generation output file header string including allele and genotype frequencies for both loci.

15. Specify output filenames and example input rows.

16. If the input file does not exist, create it with headers and example rows, then exit.

17. Read all non-empty lines from the input file.

18. If the input file is empty, repopulate it with headers and example rows, then exit.

19. Verify that the first line matches the expected header; exit if it does not.

20. If only the header is present, append example rows and exit.

21. Validate each data line to ensure it contains exactly ten semicolon-separated fields.

22. Validate that initial population size (Ni) is a positive integer within allowed bounds.

23. Validate that growth rate (r) is greater than –1.

24. Validate that carrying capacity (K) is an integer ≥ Ni.

25. Validate that selection coefficient s\_A is a float in [–2, 2].

26. Validate that the number of attempts is a positive integer within allowed bounds.

27. Validate that dominance coefficient h\_A is a float in [–1, 1].

28. Validate that initial allele frequency p\_A\_i is in [0, 1].

29. Validate that selection coefficient s\_B is a float in [–2, 2].

30. Validate that dominance coefficient h\_B is a float in [–1, 1].

31. Validate that initial allele frequency p\_B\_i is in [0, 1].

32. Store valid parameter sets in a list; exit if any validation errors occurred.

33. Define the simulate\_population function to model two independent loci under selection, drift, and density-dependent population dynamics.

34. Initialize counters and accumulators for fixation events, population sizes, and pan-homozygous states for both loci.

35. Precompute fitness values for all genotypes at both loci using s\_A, h\_A, s\_B, and h\_B.

36. Compute Beverton-Holt constants for population growth modeling.

37. If per-generation output is enabled, open a temporary file for writing per-generation data.

38. For each simulation attempt, initialize population size and allele frequencies for both loci.

39. For each generation, compute and write current state (N, allele frequencies, heterozygosity) to the temporary per-generation file if enabled.

40. After writing, check for fixation (p ≥ 1 or p ≤ 0) at each locus and record fixation status, generation, and population size.

41. Detect when pan-heterozygosity is lost (any locus fixed) and record that generation and population size.

42. Terminate the attempt when both loci have reached fixation (pan-homozygous state) and update all relevant accumulators.

43. Update population size each generation using Beverton-Holt (r ≥ 0) or exponential decay (r < 0), with stochastic rounding.

44. If population size drops to zero, declare both un-fixed alleles lost and record extinction generation.

45. Update allele frequencies for each unfixed locus using selection (fitness-weighted) followed by binomial sampling (drift).

46. Close the temporary per-generation file if it was opened.

47. Compute and return average fixation statistics for both loci, pan-homozygous metrics, and the temporary filename.

48. Define a worker function that sets a unique random seed per (SimNr, Rep) and calls simulate\_population.

49. In the main block, determine the number of available CPU cores.

50. Overwrite the detailed results file with its header.

51. Overwrite the averaged results file with its header.

52. If per-generation output is enabled, initialize the per-generation file with its header.

53. Loop over each valid parameter set (SimNr).

54. For each SimNr, create a job for every repetition.

55. Execute all repetitions for the current SimNr in parallel using multiprocessing.

56. Sort the results by repetition number to ensure consistent output order.

57. If per-generation output is enabled, merge temporary per-generation files into the master per-generation results file in repetition order.

58. Delete each temporary per-generation file after merging.

59. Write individual repetition results (fixation probabilities, generations, population sizes) to the detailed results file.

60. Compute derived metrics for heterozygote loss at each locus as weighted averages of fixation outcomes.

61. Compute averaged results across all repetitions for the current SimNr using nanmean to handle missing values.

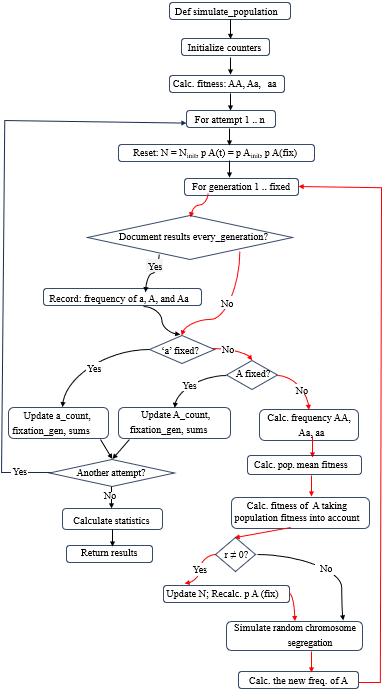
62. Adjust heterozygote loss metrics to avoid unjustified NaNs when fixation probability is exactly 1.

63. Write the averaged results for the current SimNr to the averaged results file.

64. Print confirmation messages indicating where output files were saved.

65. Print the current working directory.

66. Print the total program execution time.



**Figure 1**. Flow logic of the key function *simulate\_population* in fix\_1\_haplo.py