**Programming process steps in fix\_2\_haplos.py**

**Overview**

This Python program **simulates the probability of fixation and the number of generations required for alleles in two unlinked genes (A and B) in a diploid organism**. The program can be used to test assumptions relevant to evolutionary and young earth creationist models, specifically concerning **initial allele proportions** (pAi​​ and pBi​​) and **selection coefficients** (sA​ and sB​).

**Overview and assumptions**

* **Wright-Fisher Model**: It assumes **random mating**, where allele frequencies in the next generation are sampled from a binomial distribution based on the current generation's effective population size and allele frequencies.
* **Population Growth**: The population size (N) changes over generations according to the **discrete Beverton-Holt model**, using an intrinsic growth rate (r) and a carrying capacity (K). This allows constant, increasing, or decreasing population sizes.
* **Two Unlinked Genes**: The simulation tracks two separate genes, A and B, located on different chromosomes, meaning they segregate independently. Each gene has two alleles (e.g., A/a and B/b).
* **Selection and Dominance**: The fitness of genotypes for each gene is determined by a **selection coefficient** (s) and a **dominance coefficient** (h). This allows for modeling advantageous, deleterious, or neutral alleles, and different modes of inheritance (recessive, additive, dominant, or over/under-dominance).
* **Fixation/Loss Tracking**: The core goal is to determine the probability of an allele (A, a, B, or b) reaching **fixation** (frequency of 1.0) or **loss** (frequency of 0.0), and the number of generations this process takes. These are determined for A, a, B, and b individually, and also for the final homozygous state (AABB, aaBB, AAbb, or aabb).
* **Multiple Attempts and Repetitions**: For each set of input parameters specified in input\_data.txt, the program runs a specified number of attempts (independent simulation runs). Furthermore, each simulation scenario (defined by a line in input\_data.txt) is also repeated multiple times using parallelization.
* **Input/Output Handling**:
  + Parameters for each simulation scenario are read from an input\_data.txt file, which is validated for correct format and values.
  + Detailed results for each individual simulation repetition are saved to results\_data.txt.
  + Averaged results across all repetitions for each parameter set are saved to results\_data\_avg.txt, providing overall statistics like average fixation probabilities and average generations to fixation, along with their standard deviations.
* **Multiprocessing**: The program utilizes Python's multiprocessing module to run multiple simulation jobs in parallel, significantly speeding up the execution of large numbers of simulations by leveraging all available CPU cores.

**Individual chronological steps executed**

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.