PROJECT FROM GROUP 13

WRIGHT-FISHER THEORY OF EVOLUTION

Alice Albrecht
Noémie Chillier
Loris Constantin
Alexia Dormann
Aïman Lavallard Fadlane

November - December 2019

Contents

1 Introduction

The aim of this program is to simulate the evolution of a genetic population over time. A population is composed of individuals, each one having a genome. At each generation, a new population of the same size is randomly chosen among individuals of the last population following the Wright-Fisher model. This model works with fixed population sizes, so that alleles compete only against other alleles and not against an external environment, and use a multibinomial distribution to define new frequencies for each allele (e.g. different genotypes). This process allows the population to evolve through generations. Moreover, we choose to improve our simulation by adding a migration feature, i.e. to move a given number of individuals between different populations.

2 Implementation

To realize this program, we used an oriented-object approach and made various tests to ensure that our programm works correctly.

2.1 Classes and functions

`Simulation` This is the most important class. It reads the inputs of the user in **main.cpp** using the TCLAP library. From this informations, it creates simulations of the same population. It then runs the program as many times as there are generetations, following the Wright-Fisher model, i.e calling the **multibinomial** function. Finally it prints each output on a file or directly on the terminal and also displays the genetic code of each population.

`Population` This class models a group of individuals with a genome, i.e number of alleles. The parameters used to create a population are given by the user in the terminal or by reading a fasta file.

`Multibinomial` This function is to calculates at each step the new frequencies of each population following the Wrigth-Fisher model. After a certain generation time all alleles will disappear except one which will reach a frequency of 1, i.e be the only one to last in the genome). This genetical event is called fixation.

`ReadFasta` The aim of this function is to read a fasta file if the user enters a file name. Reading this file with the corresponding marker sites provides the alleles and the related frequencies.

2.2 Migration extension

We choose to improve our simulation by adding the ability to move a given number of individuals between different populations. Population exchanges are determined by rates r_{ij} (fraction of population i that is moving to population j for each time step). Populations sizes are fixed, therefore number of individuals moving out of i must be equal to number moving into i for each population. The rates are implemented through an n $\times$ n matrix (n = total number of populations). Concretely we created a new class `Migration` which inherits from the class `Simulation` and performs all the migration steps, namely creating the matrices for the different migration modelisations (ring, star or complete migration), compute the exact number of individuals to move (using uniform distribution to determine the ratios) and modify the different populations accordingly.

2.3 Tests

In order to control the behaviour of our programm and improve it efficently we used Google Test. Gtest is a unit testing library for C++ allowing to control specif feature of our code. The tests can be run one at a time, or be called to run all at once. This makes the debugging process very specific. To execute the tests, type the command: `make test` or `./testWrightFisher`.

For Multibinomial

`somme1`	Checks if the sum of the alleles' frequencies remains equal to 1 after a given number of calls to multibinomial (10 in this case).
`fixation`	Controls that if an allele disappear (i.e. its frequence is null) it won't reappear again in the simulation (corresponds to extinction of the character).

For Simulation

`sameAverage`	Tests the stability of alleles frequencies over time. For a large number of populations, frequencies after 1, 2, 3 generations on average are the same as for $t = 0$.
`fixationTime`	For any simulation, one allele will always reach a frequency equal to 1 (the only one to remain). This test verify it's the case (tested here with 2 and 5 alleles).
`outputFile`	Checks if the output corresponds to the expected values.

For Migration

`createMatrix`	Verifies that the sum of the ratios on line i is equal to the sum of the ratios on column i and that both sums are less or equal than 1 (tested here with a population of 10 individuals).
`popNumType`	Controls the stability of the populations size after performing a migration. Type can be "Complete", "Star" or "Ring". The size of a given population should be the same before and after all migration steps (tested here for 20 populations of 200 individuals each).
`sumOfFreq`	Check if the sum is still equal to 1 after migration
`OnlyMigrate`	Tests only the extension Migration without using mulibinomial function.

For ReadFasta

`readNormalFasta`	Controls the reading of the file test.fa (which as the expected formatting).
`readBadlyWrittenFasta`	Controls correct reading of the file test_bad.fa, containing different format anomalies. It still needs to be understood by ReadFasta.
`sort`	Verifies that the alleles are sorted alphabetically when a fasta file is read.

3 Use of the Program

To run the simulation, we can either give arguments in the terminal or read them from a fasta file, provided by the user with `-F` or `-File`. To get a overview of all the parameters the user can enter `-h` or `-help`.

3.1 From the terminal

`-N` Specify a positive number that will represent the size of the population, i.e: number of individuals per population. `DEFAULT: 100`

`-G` Positive number representing the duration of the simulation, i.e. the number of generations. `DEFAULT: 10`

`-f` Positive floating number between 0 and 1 representing the list of the initial alleles' frequencies of the population. `NOTE:` there must be as many frequencies as there are alleles. The sum of all the frequencies must be equal to 1.

`-R` Positive number representing the number of times the simulation is repeated with the same parameters. `DEFAULT: 3`

`-M` String representing the migration type (star, ring or complete).

`-P` Boolean which indicates if the output must be in a file.

`-T` Boolean which indicates if the output must be in terminal.
`NOTE:` It can be print in terminal and/or in file.

In the case the user does not want to work, with fasta file we put default values for each parameters, expect for the frequencies **-f** and the output type that the user must specify.

3.2 From a Fasta File

`-F` String representing the name of the fasta file.
`NOTE:` the fasta file must contain nucleotides sequences composed of A, T, C, G or N. In this last case, N will be replaced randomly by another nucleotide. The other lines (chromosome number) have to start with the symbol "`>`".
Example:

```
>chr11
CACTGTGTNTGATGATCAACAAAAACCTGGGGGGGGTAGTAGTAGTCC
>chr7
AAATGGTGCGTGATGCCCCCCCCCCCCCNCCTTGTGAAAA
```

`-m` Positive number representing the list of markers, i.e. the position of the nucleotide of each allele in the fasta file.

Same as in the section *From the terminal* above: **-G, -M, -R, -P, -T**. We put default values for each parameters, expect for the fasta file **-F** and the markers **-m**. The user has to provide at least the fasta file, the desired markers and the type of output (terminal and/or file).

4 Examples of simulations

- Execution of the program with parameters in the terminal (whitout fasta file):
  `./WrightFisher -G 3000 -R 500 -N 5000 -f 0.6 -f 0.14 -f 0.13 -f 0.13 -P`
  Runs a simulation on 3000 generations of 500 populations with 5000 individuals and 4 alleles of frequencies 0.6, 0.14, 0.13, 0.14 (sum equal 1) and prints only in file named "test.txt", which is in the folder build. `EXECUTION TIME:9.5s`. If we add `-T` it also prints in the terminal. `EXECUTION TIME:30.2s`.

- Execution of the program when we read a long fasta file:
  `./WrightFisher -F ../database/mitogene.fa -G 3000 -R 500 -m 190 -m 191 -m 192 -m 193 -m 194 -m 195 -m 196 -m 197 -m 198 -m 199 -m 200 -m 201 -m 202 -m 203 -m 204 -m 205 -m 206 -m 207 -m 208 -m 209 -m 210 -P`
  Runs a simulation of 500 populations over 3'000 generations. The number of individuals, alleles and codons are deduce from the fasta file. Prints only in "test.txt", which is in the folder build. `EXECUTION TIME: 7.2s`. If we add `-T` it also print in the terminal (`EXECUTION TIME: 33.3s`).

- Execution of the program with the extension migration:
  `./WrightFisher -F ../database/test.fa -m 2 -m 4 -m 8 -R 2 -G 100 -T -P -M complete`
  Runs a simulation of 2 populations over 100 generations. The number of individuals, alleles and codons are deduced from the fasta file. Outputs are made in both terminal and file "test.txt" located in the folder build. `EXECUTION TIME: 0.1s`.

5 Interpretation

5.1 Read output

Each line bellow corresponds to one generation. The first column is the generation number (i.e. time), each subsequent column represents a replica of the simulation where each replica is represented by the frequencies of the alleles in the population (separated by '|'). If a fasta file was provided as input, the last line indicates the calculated alleles.

0	0.1|0.2|0.3|0.4	0.1|0.2|0.3|0.4	0.1|0.2|0.3|0.4
1	0.096|0.21|0.274|0.42	0.087|0.206|0.298|0.409	0.098|0.207|0.311|0.384
2	0.075|0.208|0.279|0.438	0.086|0.208|0.307|0.399	0.087|0.214|0.307|0.392
3	0.067|0.2|0.284|0.449	0.074|0.193|0.333|0.4	0.096|0.215|0.288|0.401
4	0.067|0.208|0.287|0.438	0.088|0.196|0.311|0.405	0.106|0.222|0.279|0.393
	ACG|ACA|ATA|ATG	ACT|ACA|AGA|GCA	ACG|AGA|AGT|AAA

Table 1: Standard results as displayed in terminal or text file.

5.2 Graphs

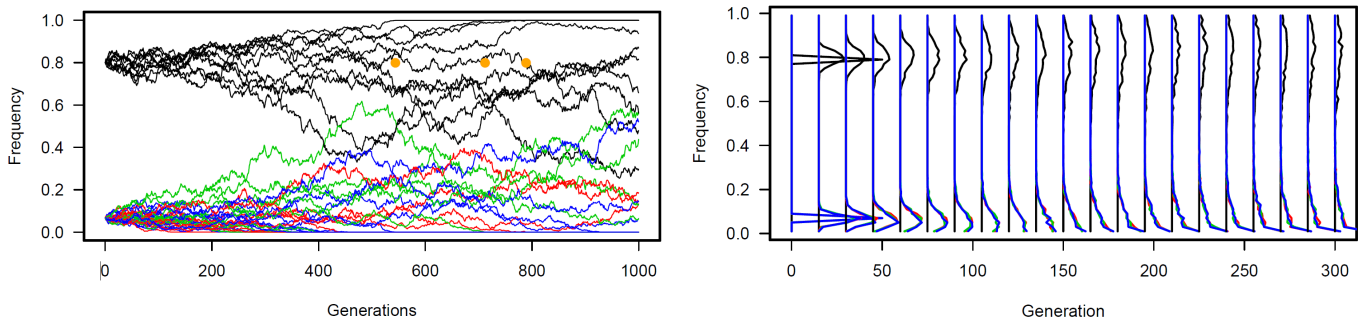

Figure 1: LEFT: Graphical representation of the fixation time for a set of simulations. The orange dots represent the fixation time (in number of generations) for each simulation. RIGHT: We can observe the staggering of the frequencies over time for different populations: they all start with the same frequencies but evolve independently. However the sum of these frequencies is stable even after hundreds of generations.