PROJECT FROM GROUP 13

WRIGHT-FISHER THEORY OF EVOLUTION

Alice Albrecht
Noémie Chillier
Loris Constantin
Alexia Dormann
Aïman Lavallard Fadlane

Date: November - December 2019

Contents

1 Introduction **2**

2 Implementation **2**
2.1 Class and function 2
2.2 Migration extension 2
2.3 Tests 3

3 Use of the Program **3**
3.1 From the terminal 3
3.2 From a Fasta File 4

4 Examples of simulations **4**

5 Interpretation **5**
5.1 Read output 5
5.2 Graphs 5

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). All this allows the population to evolve through generations. Moreover, we choose to improve our simulation by adding the ability of migration, i.e to move a given number of individuals between different populations.

2 Implementation

To realize this program, we made several classes with several links between them. We made also some tests to ensure that our programm works correctly.

2.1 Class and function

`Simulation`	This is the most important class. It reads the inputs of the user in the TCLAP in **main.cpp**. From this informations, it creates simulations of the same population. It then run the programme as many time as there are generetations, following the Wright-Fisher model, i.e calling the **multibinomial** function. Finally it prints each step of the program on a file or directly on the terminal and print also the genetic code of each population.
`Population`	This class contain a number of individus with a genome, i.e number of alleles. The parameters use to create a population are given by the user in a terminal or by reading a fasta file.
`Multibinomial`	The aim of this function is to calculate, at each step, the new frequencies of each population following the Wirth-Fisher model. With this model, we can see that 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 (fixation). In this case we can say that this allele is dominant.
`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 markers gives the alleles and the frequencies of the population.

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. Population 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 in n $\times$ n matrix (n = total number of populations).
Concretely we created a new class **migration** which herits of the class **simulation** and performs all the migration step, 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 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 (ie. 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 (so only one to remain). This test verify it's the case (tested here with 2 and 5 alleles).

For Migration

`create_matrix`	Verify 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).

For ReadFasta

`readNormalFasta`	Control the reading of the file test.fa, which is wright as expected.
`readBadlyWrittenFasta`	Control the reading of the file testbad.fa, containing different difficulties of writing, which must be manage and read by the function.
`sort`	Verify if the alleles are sort alphabetically when a fasta file is reading.

3 Use of the Program

To run the simulation, we can either gives argument from the terminal or read information from a fasta file, provide 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`	Specify a positive number that will represent the duration of the simulation, i.e: the number of generations. `DEFAULT: 10`

`-f` Specify a floating number between 0 and 1 that will represent the list of the initials frequencies of the alleles of the population. `NOTE:` there must be as many frequencies as there are alleles. The sum of all the frequencies must be equal to zero.

`-R` Specify a positive number that represent the number of time the simulation is repeat with the same parameters. `DEFAULT: 3`

`-M` Specify a string that will represent the migration type (star, ring or complete).

`-P` Specify a boolean that will indicate if the output must be in a file.

`-T` Specify a boolean that will indicate 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 so the user have to enter at least the frequencies and where he wants to print the output.

3.2 From a Fasta File

`-F` Specify a string that represent 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 replace randomly by another nucleotide. The other lines (chromosome number) have to start with ">" symbol.

Example:

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

`-m` Specify a positive number that will represent 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 name of the fasta file **-F** and the markers **-m**, so the user have to provide at least the name of the fasta file, the markers corresponding to read it and where he wants to print the output.

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
  ```
  Run 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 print 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
```
Run a simulation on 3000 generations of 500 populations. The number of individuals, alleles and codons are deduce from the fasta file. Only print 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
  ```
  Run a simulation on 100 generations of 2 populations. The number of individuals, alleles and codons are deduce from the fasta file. Since we put 4 markers, there will be 5-4 alleles. The migration will be complte. It then and prints in terminal and in file named "test.txt", that you can find in the folder build.`EXECUTION TIME: 0,0s`.

5 Interpretation

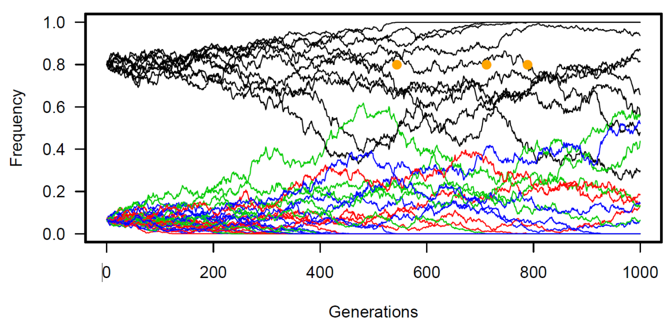
5.1 Read output

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 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



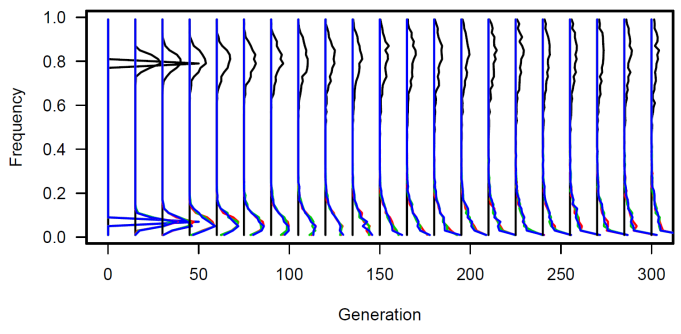


Figure 1: LEFT Graphical representation of the fixation time for a set of simulations. The orange dots represent the fixation time (in number of generations) for each simulation. RIGHT We can observe the staggering of the frequencies over time for different populations: they all start with the same frequencies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