Computer exercise 1 – Genetic drift and selection

Martin Ryberg

November 13, 2015

In this exercise you will simulate gene frequencies in small populations. The populations will consist of diploid hermaphroditic individuals that are obligate outcrossers. This mean that each individual will have two copies of the gene, and be able to act like either male or female, but not fertilize itself, i.e. each offspring will have two separate parents randomly selected among the individuals in the population.

The simulations will be done in R and the population will be represented as a list with a bin for each individual. Each individual (list bin) will have a vector with two bins, one for each copy of the gene. The different alleles will be represented with integer numbers. In the examples below you will work with two alleles, i.e. 1, 2, but it is possible to introduce more alleles.

To help you simulate the evolution you have a number of functions in the file "exercise2functions.r". You can read these functions into R using the source() function. If you have downloaded the file and placed it in the working directory of R you may use this command:

```
source("exercise2functions.r")
```

The three main functions are create_population(), new_generation(), and introduce_mutation(). Then there are several functions to get different statistics from the population and to help the main functions. create_population() will create a start population with each allele randomly drawn for each individual. new_generation() will return a population as large as the population passed to it. Each individual in the new population will have two parents drawn from the population passed to the function, and one gene copy randomly drawn from each parent. The probability of each individual in the population passed to the function to be drawn as a parent is proportional to its relative fitness (passed as a separate argument). The default is to give each individual the same relative fitness. introduce_mutation() will randomly choose an individual, and randomly choose one of its gene copies and assign it as a new allele. The new allele will be represented by the highest number used for any allele in the population plus one. So if the highest integer used to identify an allele in the population is 1 then the mutant allele will be identified by the number 2, if the highest number is 2 the mutant allele will be identified by the number 3. If you want to check the code of any function you can just write the function name in R, or open the source code file in a text editor.

Many of the simulations will be rather slow, but try to do at least 10 repeats to get a decent sample. You may also look at what the others in the group find, so you will get a larger sample.

First you will simulate genetic drift. That is neutral evolution (i.e. changes in gene frequencies) in a small population. Set the population size to 100. Keep this in a variable so you easily can reuse code, even if you change the population size. Then create a population using create_population(). The default is to create a diploid population with two alleles so you only need to pass the population size as an argument. Create a vector to store the number of copies of each allele in the population and use the function allele_numbers() to get the number for the two alleles in the start population. The allele_numbers() function takes the population as one argument and the number of alleles as second argument. Also create a variable to store how many generations have been simulated and assign it zero as starting value.

```
population_size<-100
my_population<-create_population(population_size)
numbers_of_alleles<-allele_numbers(my_population,2)
n<-0
```

Once this is set up you are ready to simulate the evolution. The n_haplotypes() function will return how many different alleles there are in the population. Continue the simulation until there is only one allele left, which means that that allele is fixed. In each iteration the population will be replaced by the next generation. This is the same as not having any overlap in generations, that is, all individuals in one generation dies and the next generation takes over. Add a generation to the count, and add the numbers of each allele to the vector for this.

```
while(n_haplotypes(my_population)>1) {
    new_generation(my_population)->my_population
    n<-n+1
    numbers_of_alleles<-c(numbers_of_alleles,
        allele_numbers(my_population,2))
}
```

You can check how many generations it took to fixation by printing the variable counting this.

```
n
```

You can check which allele that was fixed by checking any of the individuals in the population.

```
my_population[[1]]
```

To be able to easily plot the allele frequency of the fixed allele, you can convert the vector of allele numbers to a matrix where each allele is in its own row and each generation in its own column.

```
numbers_of_alleles<-matrix(numbers_of_alleles,nrow=2)
```

You can then plot it with:

```
plot(numbers_of_alleles[my_population[[1]][1],]/(2*population_size),
    type='l',ylim=c(0,1),xlab="Generations",ylab="Frequency",
    main='Two alleles, population size 100',col='red')
```

How large is the variation in how many generations it takes for an allele to be fixed? Which allele is fixed?

Even if I do not repeat it below it may be worth plotting the changes in gene frequencies over the generations in selected simulations below, just to better understand what is happening.

Next you should see what happens when you introduce some selection. You can give the new_generation() function a vector where each bin has the name of the possible allele combinations, e.g. '1,1', '1,2', '2,2', and the relative fitness (the numbers will be relative to each other), e.g. 1.0, 1.1, 1.1. The function get_named_vector() will create the named vector for you but you need to assign the values.

```
my_population<-create_population(population_size)
rel_fitness<-get_named_vector(2)
rel_fitness[1:3]<-c(1,1.1,1.1) # 2 is a dominant allele
numbers_of_alleles<-allele_numbers(my_population,2)
n<-0
while(n_haplotypes(my_population)>1) {
    new_generation(my_population,rel_fitness)->my_population
    n<-n+1
    numbers_of_alleles<-c(numbers_of_alleles,
        allele_numbers(my_population,2))
}
# Check number of generations to fixation
n
# Check which allele is fixed
my_population[[1]]
```

What happens to the number of generations to fixation? Which allele is fixed?

In the above simulation the selection for allele 2 is very strong. Repeat the simulation above but set a lower relative fitness for allele 2.

```
rel_fitness[1:3]<-c(1,1.01,1.01) # dominant allele
```

How does that affect the results?

Now increase the population size to 200, but keep the lower selective advantage of allele 2.

```
population_size<-200
```

What happened?

Next you will investigate how large chance a new mutation has to make it into the population? Reset the population size to 100. Now you should start with a population that is fixed for one allele, so add the haplotypes=1 argument to the call to the create_population() function. You then use introduce_mutation() function to introduce a new allele. The new allele will only be present as one copy in one allele and will be identified by the number 2. These simulations are fast so do many 30-50 repeats. It may be worth plotting a few (but far from all).

```
population_size<-100
my_population<-create_population(population_size,haplotypes=1)
```

```
my_population<-introduce_mutation(my_population)
numbers_of_alleles<-allele_numbers(my_population,2)
n<-0
while(n_haplotypes(my_population)>1) {
    new_generation(my_population)->my_population
    n<-n+1
    numbers_of_alleles<-c(numbers_of_alleles,
        allele_numbers(my_population,2))
}
numbers_of_alleles<-matrix(numbers_of_alleles,nrow=2)
n
my_population[[1]]
```

What do you observe? Try with increased population size again.

```
population_size<-200
```

Continue to increase the population size in steps of 100 up to 500, and repeat the simulations for each population size. When the population size get larger some simulations may become a bit slow.

Try also to decrease the population size and see what happens.

```
population_size<-50
```

Repeat the simulations for decreasing population sizes in steps of 10 to final population size of 10.

What happens? Now repeat the simmulations with strong selection.

```
population_size<-100
my_population<-create_population(population_size,haplotypes=1)
my_population<-introduce_mutation(my_population)
rel_fitness<-get_named_vector(2)
rel_fitness[1:3]<-c(1,1.1,1.1)
numbers_of_alleles<-allele_numbers(my_population,2)
n<-0
while(n_haplotypes(my_population)>1) {
    new_generation(my_population,rel_fitness)->my_population
    n<-n+1
    numbers_of_alleles<-c(numbers_of_alleles,
        allele_numbers(my_population,2))
}
numbers_of_alleles<-matrix(numbers_of_alleles,nrow=2)
n
my_population[[1]]
```

Try what happens now if you increase population size?

```
population_size<-200
```

Try to increase the population size in steps of 100 up to 500. Again, here the simulations may become a bit slower.

And again, what happen if you decrease population size?

```
population_size<-50
```

Try with decreasing population sizes in steps of 10 down to 10.

What do you think will happen if you decrease the relative fitness of '1,2' and '2,2'? Do you need to make simulations to feel confident in your answer?

Change the relative fitness so it is an recessive allele, change population size to 200, and repeat the simulation.

```
population_size<-200
rel_fitness[1:3]<-c(1,1,1.1)
```

What happened?

The report should be one to two paragraphs long and account for your reflections on genetic drift and selection based on your observations from the simulations.