

Simulating DNA

Athens COST InsectIMP Course 2025

Jana Obšteter, Alireza Ehsani, and Gregor Gorjanc

2025-01-30

Introduction

The DNA molecule consists of four bases: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). It is formed as a double helix in which the bases bind in pairs: Adenine (A) binds with Thymine (T) and Cytosine (C) binds with Guanine (G). All DNA molecules of an organism form its genome. The structure and size of the genome varies between species. For example, we humans have our genome organised into 23 chromosomes (22 autosomal chromosomes and 1 sex chromosome). Each chromosome is present in two copies (we are diploid organisms). We get one copy of chromosomes from each of the two parents. We call each chromosome copy a haplotype and the combination of two copies, a genotype. The total length of our genome is about 3 billion base pairs (this is the length of one copy of chromosomes - their total length spanning all 23 chromosomes). Some genomes are much smaller. For example, the honeybee genome has only about 250 million base pairs organised in 16 chromosomes. In contrast, some plant genomes are much larger. The wheat genome has about 17 billion base pairs organised in 21 chromosomes. The largest genome was found in a plant called *Paris japonica* with 150 billion base pairs!

As you could see above, genomes of different species can differ substantially because each species experienced a different demographic history. This history can lead to a different number of chromosomes, the number of chromosome copies present in the genome (ploidy), different amounts of variation between chromosome copies, etc. When running simulations, it is important to take these differences into account. **AlphaSimR** has a number of options for doing this as we will show in the following section.

Most of the genome is the same across all chromosome copies in a population. We are, however, most interested in the parts of the genome that differ between chromosome copies. This variation can be present, both within one individual, or between several individuals. These variable parts of the genome are often called loci, sites, segregating sites, or variants. These loci differ because at some point in the past one of the chromosome copies has undergone mutation and that mutation has been copied further. This means that now some chromosome copies have one base at that locus (say A) and other individuals have another base (say G). We often refer to the different versions of bases at a locus as alleles. Here we will focus only on loci with two alleles, since this type of DNA variation is the most common. We will represent this bi-allelic variation in a computer with numbers 0 and 1. The zero (0) represents an ancestral allele and the one (1) represents a mutated allele, or just mutation. A mutation is often also called a derived allele or an alternative allele.

We will now show how to start an **AlphaSimR** simulation by generating DNA (genomes) of individuals. To do this we will:

- Simulate founding genomes,
- Set global simulation parameters,
- Create a base population of individuals, and
- Look into the genomes of the base population individuals.

Founding genomes

First we clean our working environment and load the `AlphaSimR` package.

```
# Clean the working environment
rm(list = ls())

library(AlphaSimR)
```

Loading required package: R6

At the time of writing, `AlphaSimR` provides two functions for simulating genomes. The first function is `quickHaplo()`, which quickly simulates haplotypes by randomly sampling 0s and 1s. The other function is `runMacs()`, which uses method from Chen et al. (2009) to simulate haplotypes with a species-specific demography. Of note, haplotypes here are spanning whole chromosomes, so we could also call these haplotypes as whole-chromosome haplotypes or just chromosomes. Read more about the `quickHaplo()` and `runMacs()` functions in the help pages using the code below.

```
help(quickHaplo)
help(runMacs)
```

Let us now simulate founding genomes for cattle. We will use the `runMacs()` function and for simplicity and speed we will simulate only 5 individuals (`nInd = 5`), 2 chromosomes (`nChr = 2`), and capture only 4 loci per chromosome (`segSites = 4`). To be clear, cattle do not have 2 chromosome pairs. They have 30 chromosome pairs, but to speed up this session we will simulate only 2 chromosome pairs. Cattle are one of the predefined species in `runMacs()`, so we just use `species = "CATTLE"` argument. Since cattle are diploid, we will use the default `ploidy = 2` argument. This means that `runMacs()` will simulate two copies for each chromosome of each individual.

```
# Simulate a founding pool of genomes (this command takes a while to run!)
founderGenomes = runMacs(nInd = 5,
                        nChr = 2,
                        segSites = 4,
                        species = "CATTLE")
```

```
# Inspect the founderGenomes object
founderGenomes
```

```
## An object of class "MapPop"
## Ploidy: 2
## Individuals: 5
## Chromosomes: 2
## Loci: 8
```

```
str(founderGenomes)
```

```
## Formal class 'MapPop' [package "AlphaSimR"] with 8 slots
## ..@ genMap      :List of 2
## .. ..$ 1: Named num [1:4] 0 0.653 0.665 0.792
## .. ..$ 2: Named num [1:4] 0 0.145 0.44 0.577
## .. ..$ 3: Named num [1:4] 0 0.145 0.44 0.577
## .. ..$ 4: Named num [1:4] 0 0.145 0.44 0.577
## ..@ centromere: Named num [1:2] 0.396 0.289
## ..@ inbred     : logi FALSE
## ..@ nInd       : int 5
## ..@ nChr       : int 2
## ..@ ploidy     : int 2
```

```
## ..@ nLoci      : int [1:2] 4 4
## ..@ geno       :List of 2
## .. ..$ : raw [1, 1:2, 1:5] 04 0d 04 0c ...
## .. ..$ : raw [1, 1:2, 1:5] 01 04 06 04 ...
```

Global simulation parameters

After creating the founding genomes, we must use the `SimParam()` function to create an object that contains global simulation parameters. These simulation parameters will be common to all downstream AlphaSimR work. You can read more about the `SimParam()` function and its *many* methods in the help page provided with `help(SimParam)`.

We will create a `SimParam` object named `SP` based on the founding genomes. When we name the `SimParam` object as `SP`, AlphaSimR functions recognise it and use it automatically without you having to pass it as an argument to each function.

```
# Create an object holding the simulation parameters
SP = SimParam$new(founderGenomes)
#str(SP)
```

Since sex is not assigned to individuals by default, we will request this through `SimParam$setSexes()` function. Since the simulation parameters will be stored in the `SP` object, we actually need to call `SP$setSexes()`. We will use `yes_sys`, option, which will systematically create a male and a female in a sequence (male, female, male, female, etc.). Read about the other sex options with `help(SimParam)` (scroll down to `SimParam$setSexes()`).

```
# Set sex of newly created individuals systematically
SP$setSexes("yes_sys")
```

A base population of individuals

Now we can use the `newPop()` function to create a population (a group) of individuals. Genomes of these individuals will be created from the founding genomes, while their sex will be allocated according to `SP$setSexes()` (see `help(newPop)` for more information). Let's name this population as a base population or simply `basePop`.

```
# Create a population of individuals from the founderGenomes
basePop = newPop(founderGenomes)
```

```
# Inspect the basePop object
basePop
```

```
## An object of class "Pop"
## Ploidy: 2
## Individuals: 5
## Chromosomes: 2
## Loci: 8
## Traits: 0
```

AlphaSimR populations contain information about its individuals - their identification (name), their mother and father, their sex, information about the traits (we will talk about traits later), miscellaneous information, the number of individuals and genome information (the number of chromosomes, ploidy, and the number of loci) and finally genomes of the population individuals.

```
# Inspect the structure of the population object
# (there will be lots of output of which some will be "empty" - we will populate
# some of it in next sessions.)
str(basePop)
```

```
## Formal class 'Pop' [package "AlphaSimR"] with 18 slots
##   ..@ id      : chr [1:5] "1" "2" "3" "4" ...
##   ..@ iid     : int [1:5] 1 2 3 4 5
##   ..@ mother  : chr [1:5] "0" "0" "0" "0" ...
##   ..@ father  : chr [1:5] "0" "0" "0" "0" ...
##   ..@ sex     : chr [1:5] "M" "F" "M" "F" ...
##   ..@ nTraits: int 0
##   ..@ gv      : num[1:5, 0 ]
##   ..- attr(*, "dimnames")=List of 2
##   .. .. $ : NULL
##   .. .. $ : NULL
##   ..@ pheno   : num[1:5, 0 ]
##   ..- attr(*, "dimnames")=List of 2
##   .. .. $ : NULL
##   .. .. $ : NULL
##   ..@ ebv     : num[1:5, 0 ]
##   ..@ gxe     : list()
##   ..@ fixEff  : int [1:5] 1 1 1 1 1
##   ..@ misc    : list()
##   ..@ miscPop: list()
##   ..@ nInd    : int 5
##   ..@ nChr    : int 2
##   ..@ ploidy  : int 2
##   ..@ nLoci   : int [1:2] 4 4
##   ..@ geno    :List of 2
##   .. .. $ : raw [1, 1:2, 1:5] 04 0d 04 0c ...
##   .. .. $ : raw [1, 1:2, 1:5] 01 04 06 04 ...
```

R objects (such as `founderGenomes`, `basePop`, `SP`, and others) have different structure and contents. To work with such different objects, R uses object oriented programming. There are different styles of object oriented programming in R. Most R objects use the S3 style, where elements of an object are accessed via `object$element` (note the `$`). AlphaSimR uses the S4 style for population objects, where elements of an object are accessed via `object@element` (note the `@`). We will now inspect contents of the `basePop` object.

```
# Inspect the number of individuals
basePop@nInd
```

```
## [1] 5
```

```
# Inspect the individuals' identifications (their "name")
basePop@id
```

```
## [1] "1" "2" "3" "4" "5"
```

```
# Inspect the individuals' sex
basePop@sex
```

```
## [1] "M" "F" "M" "F" "M"
```

```
# Inspect the number of chromosomes simulated
basePop@nChr
```

```
## [1] 2
```

```
# Inspect the number of loci per chromosome
basePop@nLoci
```

```
## [1] 4 4
```

Genomes of population individuals

We can also access the genome of these individuals. We use the `pullSegSiteHaplo()` function to get haplotypes across all segregating sites (loci). See `help(pullSegSiteHaplo)` for more options on extracting haplotypes.

```
# Extract the haplotypes of individuals
```

```
pullSegSiteHaplo(basePop)
```

```
##      1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
## 1_1    0  0  1  0  1  0  0  0
## 1_2    1  0  1  1  0  0  1  0
## 2_1    0  0  1  0  0  1  1  0
## 2_2    0  0  1  1  0  0  1  0
## 3_1    0  0  1  1  0  0  0  0
## 3_2    0  0  0  1  0  0  1  0
## 4_1    1  0  1  1  0  0  1  0
## 4_2    0  1  1  0  0  0  1  0
## 5_1    0  0  1  0  0  0  0  1
## 5_2    1  0  1  1  0  0  1  0
```

The above output is a matrix with haplotypes stored in rows and locus alleles carried by the haplotypes stored in columns. Columns span across all chromosomes. Since we simulated a diploid species, we get two haplotypes (rows) per individual. Haplotypes (rows) are named as individual_haplotype, that is, 1_1 and 1_2 for the individual 1 and its two haplotypes. Loci (columns) are named as chromosome_locus, that is 1_1 and 1_2 for the chromosome 1 and its first and second locus.

The combination of haplotypes of an individual form the genotype of that individual. When we present a genotype as allele dosage (this is the number of mutations that an individual carries at each locus) we can obtain individuals' genotype by simply adding up its haplotypes. We can combine `AlphaSimR` and other R functionality to obtain the genotype of an individual as shown.

```
# Extract the haplotypes and save them into object hap
```

```
hap = pullSegSiteHaplo(basePop)
```

```
hap
```

```
##      1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
## 1_1    0  0  1  0  1  0  0  0
## 1_2    1  0  1  1  0  0  1  0
## 2_1    0  0  1  0  0  1  1  0
## 2_2    0  0  1  1  0  0  1  0
## 3_1    0  0  1  1  0  0  0  0
## 3_2    0  0  0  1  0  0  1  0
## 4_1    1  0  1  1  0  0  1  0
## 4_2    0  1  1  0  0  0  1  0
## 5_1    0  0  1  0  0  0  0  1
## 5_2    1  0  1  1  0  0  1  0
```

```
# Inspect the first haplotype of the first individual
```

```
hap[1, ]
```

```
## 1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
##   0  0  1  0  1  0  0  0
```

```
# Inspect the second haplotype of the first individual
```

```
hap[2, ]
```

```
## 1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
```

```
## 1 0 1 1 0 0 1 0
# Add up the haplotypes to get genotype
hap[1, ] + hap[2, ]
```

```
## 1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
## 1 0 2 1 1 0 1 0
```

Because accessing the genotype of an individual is a common operation, we also have the `pullSegSiteGeno()` function. This function returns one row per individual, which is the sum of haplotype rows as shown above.

```
# Extract the genotypes of individuals
pullSegSiteGeno(basePop)
```

```
## 1_1 1_2 1_3 1_4 2_1 2_2 2_3 2_4
## 1 1 0 2 1 1 0 1 0
## 2 0 0 2 1 0 1 2 0
## 3 0 0 1 2 0 0 1 0
## 4 1 1 2 1 0 0 2 0
## 5 1 0 2 1 0 0 1 1
```

There are other functions to access genomic information from **AlphaSimR** population objects. You can read about these functions from the list of all **AlphaSimR** functions provided by `help(package = "AlphaSimR")`.

We should point out, that up to now all segregating loci are neutral in the sense that they do not affect any trait. We will add causal loci and traits to our simulation in another session.

References

Chen G.K., Marjoram P., Wall J.D. (2009) Fast and flexible simulation of DNA sequence data. *Genome Research*, 19:1, 136–142. <https://doi.org/10.1101/gr.083634.108>