MetaPopGen Tutorial 3

Marco Andrello

July 30, 2020

Contents

1 Introduction

MetaPopGen 2.0 can be used to simulate multi-locus genetic processes. This vignette explains how to set up and run multi-locus simulations.

To cite the multilocus version of MetaPopGen, please use:
Andrello M et al. (2019). A multi-locus demo-genetic simulator to model populations of large sizes. **Molecular Ecology Resources**, submitted.

2 Installation

The easiest way to install MetaPopGen 2.0 is through devtools. After installing the devtools package in R, run the followin code

```
devtools::install_github(MarcoAndrello/MetaPopGen-2.0)
```

You may want to replace the "2.0" with the latest version of the package. Check on https://github.com/MarcoAndrello/

Once MetaPopGen 2.0 is installed, load it:

```
library(MetaPopGen)
```

To read the help files and see how the program works, you can type :

```
?MetaPopGen
```

3 Initializing the simulations

3.1 Defining the simulation parameters and running the initialization function

The first piece of information needed to initialize a new simulation is a vector containing the **number of alleles** at each locus. For example, to simulate two loci with two alleles each, type:

```
allele_vec <- c(2,2)
```

Next you need to define the other parameters of the simulation, namely:
- The **recombination rate** r;
- The **mutation rate** for each locus mu;
- The **number of demes** n;
- The **number of age-classes** z;
- The **carrying capacity** of each deme $kappa0$;
- The **sexuality** of the species *sexuality*, either "monoecious" or "dioecious";

These parameters have to be used as arguments in the function initialize.multilocus:

```
init.par <- initialize.multilocus(allele_vec=allele_vec,
                                  r=0.5,
                                  mu=c(0.01,0.01),
                                  n=3,
                                  z=1,
                                  kappa0=100,
                                  sexuality="monoecious")
```

```
## Generating genotype index
## Generating meiosis matrix
## Generating genotype mapping
## Generating N1
## Done
```

The code shown above initializes the simulations for a monoecious species with two loci with two alleles each, a recombination rate $r = 0.5$, a mutation rate $mu = 0.01$ at each locus, $n = 3$ demes, $z = 1$ age class and a carrying capacity $kappa0 = 100$ individuals per deme. The function initialize.monoecious defines the genetic composition of the **initial population** *N1* (or *N1_F* and *N1_M* in the case of dioecious life-cycles) by assigning an equal number of individuals to each genotype and each age-class across all demes, and stores it in the output init.par, along with other parameters needed for the simulation.

```
names(init.par)
```

```
##  [1] "allele_vec"                "r"
##  [3] "mu"                        "m"
##  [5] "n"                         "z"
##  [7] "kappa0"                    "index_matr"
##  [9] "meiosis_matrix"            "mat_geno_to_index_mapping"
## [11] "N1"                        "method"
```

```
init.par$N1
```

```
## , , age = 1
##
##            deme
## genotype     1  2  3
##   A1B1/A1B1 11 11 11
##   A1B1/A1B2 11 11 11
##   A1B2/A1B2 11 11 11
##   A1B1/A2B1 11 11 11
```

```
##   A1B1/A2B2 11 11 11
##   A1B2/A2B2 11 11 11
##   A2B1/A2B1 11 11 11
##   A2B1/A2B2 11 11 11
##   A2B2/A2B2 11 11 11
```

This shows that there are nine genotypes. We see that there are 11 individuals per genotype per age-class in each deme. This is because the function defines the number of individuals so that the sum over genotypes and age-classes in each deme roughly equals the carrying capacity *kappa0* of the deme. The initial composition can be changed by reassigning the elements of init.par$N1. The following code results in the first and second deme containing only A1B1/A1B1 individuals and the third deme containing only A2B2/A2B2 individuals.

```
init.par$N1[,,] <- 0
init.par$N1[1,c(1,2),1] <- 100
init.par$N1[9,3,1] <- 100
init.par$N1
```

```
## , , age = 1
##
##            deme
## genotype      1   2   3
##   A1B1/A1B1 100 100   0
##   A1B1/A1B2   0   0   0
##   A1B2/A1B2   0   0   0
##   A1B1/A2B1   0   0   0
##   A1B1/A2B2   0   0   0
##   A1B2/A2B2   0   0   0
##   A2B1/A2B1   0   0   0
##   A2B1/A2B2   0   0   0
##   A2B2/A2B2   0   0 100
```

Lastly, we need to define the **simulation time** *T_max*; here, we use 5 time steps

```
T_max <- 5
```

3.2 Parametrizing survival probabilities

Survival probabilities can be dependent on genotype, age-class, deme, time and, in the case of dioecious life-cycle, sex. The easiest way to define survival probabilities is to create an array of dimensions m (number of genotypes), n (number of demes), z (number of age-classes) and *T_max* (simulation time) and fill it with the desired survival probabilities. Here, we simply assign the same survival probability to each class:

```
sigma <- array(0.75,c(init.par$m, init.par$n, init.par$z, T_max))
sigma[,,,1]
```

```
##       [,1] [,2] [,3]
##  [1,] 0.75 0.75 0.75
##  [2,] 0.75 0.75 0.75
##  [3,] 0.75 0.75 0.75
##  [4,] 0.75 0.75 0.75
##  [5,] 0.75 0.75 0.75
##  [6,] 0.75 0.75 0.75
##  [7,] 0.75 0.75 0.75
##  [8,] 0.75 0.75 0.75
##  [9,] 0.75 0.75 0.75
```

However, this does not define the names of the dimensions as in init.par$N1. It is practical to have dimension

names to quickly identify to which genotype, deme, etc. an entry refers. A smart way to have dimension names without creating them by hand is to copy them from init.par$N1. We still need to name the fourth dimension (time), but that is relatively easy.

```
name.dim <- dimnames(init.par$N1)
name.dim$time <- c(1:T_max)
dimnames(sigma) <- name.dim
sigma[,,,1]
```

```
##            deme
## genotype       1    2    3
##   A1B1/A1B1 0.75 0.75 0.75
##   A1B1/A1B2 0.75 0.75 0.75
##   A1B2/A1B2 0.75 0.75 0.75
##   A1B1/A2B1 0.75 0.75 0.75
##   A1B1/A2B2 0.75 0.75 0.75
##   A1B2/A2B2 0.75 0.75 0.75
##   A2B1/A2B1 0.75 0.75 0.75
##   A2B1/A2B2 0.75 0.75 0.75
##   A2B2/A2B2 0.75 0.75 0.75
```

3.3 Parametrizing fecundities

Female and male fecundities can be parametrized with the same procedure used for survival probabilities, i.e by creating an array with the good dimensions, filling it with the desired values and then copying the dimension names from init.par$N1.

```
phi_F <- array(30,c(init.par$m, init.par$n, init.par$z, T_max))
phi_M <- array(100,c(init.par$m, init.par$n, init.par$z, T_max))
dimnames(phi_F) <- dimnames(phi_M) <- name.dim
```

3.4 Parametrizing dispersal probabilities

Adult and propagule dispersal probabilities between demes are defined using an n x n matrix giving the probability of dispersal from column j to row i. Note that when dispersal is not symmetrical between demes, the dispersal probability from column j to row i is not the same as the dispersal probability from row i to column j. What is important is that the sum of the elements of each column cannot exceed one, while the sum of the elements of rows is unbounded. Here, we assume that adults do not move between demes:

```
delta.ad <- diag(init.par$n)
dimnames(delta.ad) <- list(destination = c(1:3),
                           origin = c(1:3))
delta.ad
```

```
##            origin
## destination 1 2 3
##           1 1 0 0
##           2 0 1 0
##           3 0 0 1
```

while propagules disperse following the matrix

```
delta.prop <- matrix(c(0.9, 0.4, 0,
                       0.1, 0.5, 0.1,
                       0,   0.1, 0.9),
                     nrow=3, ncol=3, byrow=T)
```

```
dimnames(delta.prop) <- list(destination = c(1:3),
                             origin = c(1:3))
delta.prop
```

```
##            origin
## destination   1   2   3
##           1 0.9 0.4 0.0
##           2 0.1 0.5 0.1
##           3 0.0 0.1 0.9
```

4 Performing the simulation

The simulation can be performed using the functions sim.metapopgen.monoecious.multilocus (or sim.metapopgen.dioecious.multilocus for dioecious life-cycles)

```
N <- sim.metapopgen.monoecious.multilocus(init.par=init.par,
                                          sigma=sigma,
                                          phi_F=phi_F, phi_M=phi_M,
                                          delta.prop=delta.prop, delta.ad=delta.ad,
                                          T_max=T_max)
```

```
## [1] "Augmenting delta.prop for time dimension"
## [1] "Augmenting delta.ad for age dimension"
## [1] "Augmenting delta.ad for time dimension"
## [1] "Augmenting kappa0 for time dimension"
## [1] "Initializing variables..."
## [1] "Running simulation..."
## [1] 5
## [1] "...done"
```

The results are stored in the object N. For example, look at the final composition of the population at time 5:

```
N[,,,5]
```

```
##            deme
## genotype     1  2  3
##   A1B1/A1B1 61 33  8
##   A1B1/A1B2 15 17  4
##   A1B2/A1B2  1  2  4
##   A1B1/A2B1  8 12  6
##   A1B1/A2B2  9 23 16
##   A1B2/A2B2  1  1 13
##   A2B1/A2B1  2  0  0
##   A2B1/A2B2  2  1 13
##   A2B2/A2B2  1 11 36
```

Your results will generally be different from these because the simulations are stochastic.

5 Analysing the results

MetaPopGen has several functions to analyse the results of the simulations. For example, to calculate allele frequencies, use the function *freq_alleles()*:

```
freq_alleles(N[,2,1,5], init.par)
```

```
## $counts
```

```
## $counts$LocusA
##  A1  A2
## 140  60
##
## $counts$LocusB
##  B1  B2
## 131  69
##
##
## $frequencies
## $frequencies$LocusA
##  A1  A2
## 0.7 0.3
##
## $frequencies$LocusB
##    B1    B2
## 0.655 0.345
```

Note that the function takes a vector of genotype frequencies as a first argument. Here, we have used N[,2,1,5], the vector of genotype frequencies in deme 2, age 1 at time 5. The function outputs the absolute counts and the relative frequencies of each alleles at each locus.
To compute genotype frequencies *per locus*, use the function *freq_genotypes()*:

```
freq_genotypes(N[,2,1,5], init.par)
```

```
## $counts
## $counts$LocusA
## A1A1 A1A2 A2A2
##   52   36   12
##
## $counts$LocusB
## B1B1 B1B2 B2B2
##   45   41   14
##
##
## $frequencies
## $frequencies$LocusA
## A1A1 A1A2 A2A2
## 0.52 0.36 0.12
##
## $frequencies$LocusB
## B1B1 B1B2 B2B2
## 0.45 0.41 0.14
```

Again, the argument is a vector of genotype frequencies and the output is the absolute counts and the relative frequencies of each genotype at each locus.
Genetic differentiation can be calculated using the function *fst_multilocus()*:

```
fst_multilocus(N[,,1,5], init.par)
```

```
##    LocusA    LocusB      Mean
## 0.2074762 0.2050694 0.2062542
```

Here, the argument of the function is not a vector anymore, but a 2D array. The first dimension of the array contains the genotypes while the second dimension contains the groups. In this example, the groups are the demes. The function then returns the genetic differentiation index between demes at each locus, and the mean over loci. The groups need not be demes, but can also be age classes, sexes, time steps or any

other grouping created by the user, provided that the input is a 2D vector and the groups are on the second dimension. For example, the following gives genetic differentiation between year four anf five for individuals of age one in deme two:

```
fst_multilocus(N[,2,1,4:5], init.par)
```

```
##      LocusA      LocusB        Mean
## 0.009447223 0.010795777 0.010151491
```

See also other functions to calculate gamete frequencies (*freq_gametes*), observed and expected heterozygosities (*het.obs* and *het.exp*), and linkage disequilibrium (*ld*).