MetaPopGen tutorial

Marco Andrello

July 12, 2016

Contents

1 Introduction

MetaPopGen is a forward-in-time population genetics simulator. Unlike most individual-based population genetic simulators, MetaPopGen focuses on genotype numbers rather than on individuals. Genotype numbers are iterated through time by using random number generators for appropriate probabilistic distributions to reproduce the stochasticity inherent to Mendelian segregation, survival, dispersal and reproduction.

Features included in the model are age-structure, monoecious and dioecious (or separate sexes) life-cycles, mutation, dispersal and selection. All demographic parameters can be genotype-, sex-, age-, deme- and time-dependent. MetaPopGen is therefore indicated to study large populations and very complex demographic scenarios. MetaPopGen can currently simulate only one locus.

To cite MetaPopGen, please use:
Andrello M and Manel S. (2015). MetaPopGen: an R package to simulate population genetics in large size metapopulations. **Molecular Ecology Resources**, 15: 1153-1162, DOI: 10.1111/1755-0998.12371.

2 Installation

2.1 Install all the dependencies

MetaPopGen is currently distributed as a source R package. As with all R packages, installing the package from source files requires that all the dependent libraries are installed before. MetaPopGen depends on the following libraries, which I suggest that you install before trying to install MetaPopGen:
BiasedUrn
sp
lattice
MASS
Note that these libraries may already be installed on your computer. To check if they are already installed, you can simply try to load them:

```
library(BiasedUrn)
library(sp)
library(lattice)
library(MASS)
```

If you get an error message, it means you have to install them: interactively, via the menu: Package(s), Install packages; or by typing:

```
install.packages("BiasedUrn")
install.packages("sp")
install.packages("lattice")
install.packages("MASS")
```

2.2 Download the MetaPopGen package

Download the latest MetaPopGen version, as a .tar.gz file, from:
https://sites.google.com/site/marcoandrello/metapopgen

2.3 Install and load MetaPopGen

Install MetaPopGen by typing in the R console:

```
install.packages("MetaPopGen.0.0.5.tar.gz",type="source",repos=NULL)
```

(you might have to replace the “0.0.5” with the actual version of the package)

Once MetaPopgen is installed, load it:

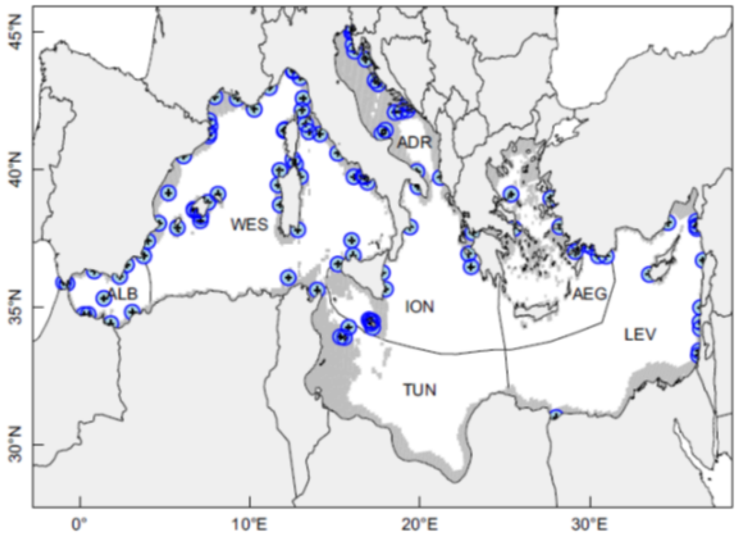
```r
library(MetaPopGen)
```

To read the help files and see how the program works (in case this tutorial is not enough!), you can type :

```r
?MetaPopGen
```

3 A simple example to get started

This is a simple example to show how MetaPopGen works. We will use an already prepared dataset for a fish population living in the Mediterranean Sea. There are 115 local populations (demes), five age-classes and sexes are separate.



We consider one locus with two alleles and the initial genotypic composition of the demes is random. Migration among demes is possible in the form of larval dispersal, and the probability of dispersal between each pair of demes was calculated using a biophysical model of larval dispersal.
The first step is loading the dataset and inspect its content:

```r
data(Mediterranean)
ls()
```

```
##  [1] "delta"     "dispersal" "kappa0"    "mu"        "N1_F"
##  [6] "N1_M"      "phi_F"     "phi_M"     "sigma_F"   "sigma_M"
## [11] "T_max"
```

These are the data you generally need to perform a simulation of population genetics with MetaPopGen. Since we are simulating a species with separate sexes, the data are often sex-specific: the objects `N1_F` and `N1_M` contain the initial genotype numbers for females and males. Similarly, `sigma_F` and `sigma_M` contain the survival probabilities; `phi_F` and `phi_M` contain the fecundities. `delta` and `dispersal` contain information on the dispersal rate, `kappa0` contain information on the carrying capacities of the demes, `mu` contains the mutation rates and `T_max` is the number of years of simulations. We will describe in deeper detail the content and the structure of these objects later in the tutorial.
We will run five replicates of each simulation, to see the variation of results due to stochasticity. In real studies, it is recommended to run a higher number of replicates, i.e. 20 or 30. The simulation results will be saved on disk, and there will be one folder for each replicate, named with the date and time of the simulation. Let's first create a folder to store the simulation results and define the number of replicates:

```r
current.dir <- getwd()
name.dir <- paste0(current.dir,"/Simulation.Med")
dir.create(name.dir)
```

```r
nrepl <- 5
```

Now, run the simulations. We use a `for` loop to run the five replicates and we use the function `sim.metapopgen.dioecious()`:

```r
setwd(name.dir)
for (i in 1 : nrepl) {

  cat("Replicate",i,"\n")

  sim.metapopgen.dioecious(
    input.type="array",
    N1_M = N1_M,
    N1_F = N1_F,
    sigma_M = sigma_M,
    sigma_F = sigma_F,
    phi_M = phi_M,
    phi_F = phi_F,
    mu = mu,
    delta = delta,
    kappa0 = kappa0,
    T_max = T_max,
    save.res = T)
}
setwd(current.dir)
```

As the simulations advance, the years will be printed on screen. The five replicate should take less than five minutes to run on most computers. In the meantime, you will see new folders being created as the simulations proceed. At the end, there will be five folders, one for each replicate, named as: YYYY-MMM-dd-hh.mm.ss (Year, Month, Day, Hour, Minutes and Seconds). If you look into each folder, you will see the simulation results, which are RData files. There is one file for each year of simulation. As we set `T_max = 200` years, there will be 200 files.
The simulation results are the number of individuals for each genotype, deme and age-class. This type of information can be used to describe the genetic structure of the population. F_{ST} is a commonly used parameter to measure genetic structure. The following code calculates the F_{ST} among all females of the 115 demes at the end of the simulation, for each replicate:

```r
list.dir <- list.dirs(recursive=F)
fst <- array(NA,dim=nrepl)
for (i in 1 : nrepl) {
  dir.name <- list.dir[i]
  fst[i] <- fst.global.dioecious(dir.name,"F",T_max)
}
```

You can see that the average F_{ST} is quite low, which is often the case for marine populations, and that there is considerable variation among replicates (your values will be different from these due to the stochasticity of the simulations) :

```r
mean(fst)
```

```
## [1] 0.007700625
```

```r
sd(fst)
```

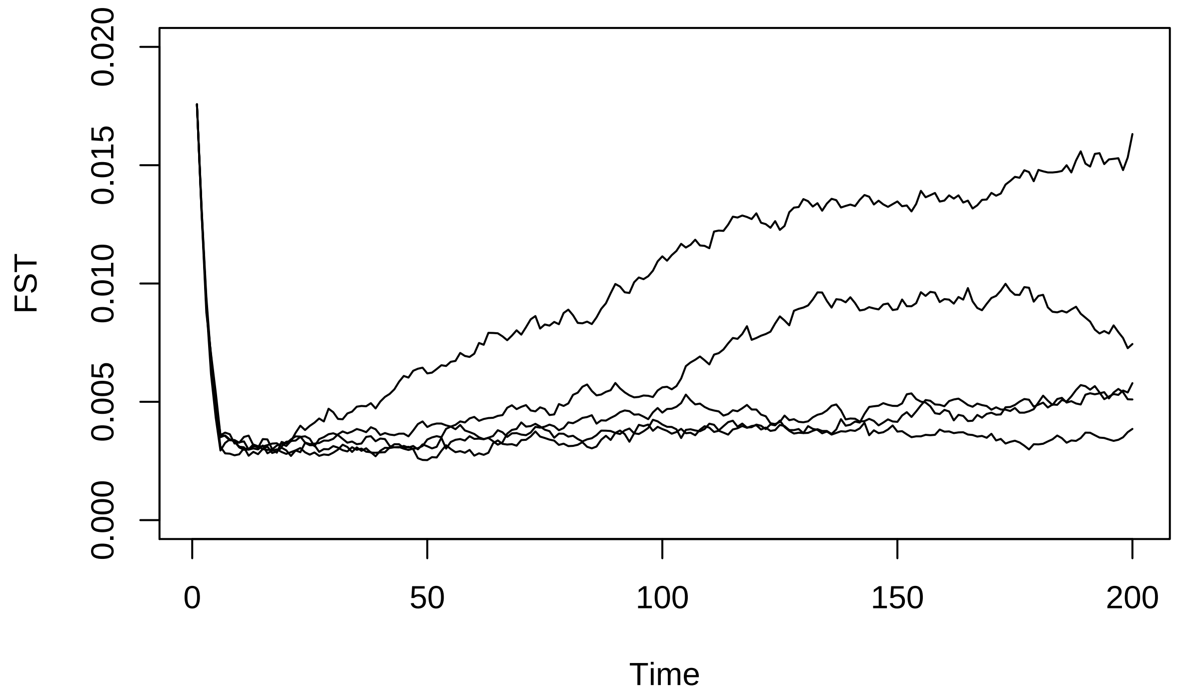
```
## [1] 0.004988
```

Since we have saved the simulation results for every year of simulation, we can look at the change in time of F_{ST} for each replicate :

```r
fst <- array(NA,dim=c(nrepl,T_max))
for (i in 1 : nrepl) {
  dir.name <- list.dir[i]
  for (t in 1 : T_max){
    fst[i,t] <- fst.global.dioecious(dir.name,"F",t)
  }
}
```

And then plot it:

```r
plot(fst[1,],ylim=c(0,0.02),type="l",xlab="Time",ylab="FST")
for(i in 2 : nrepl){
  matplot(fst[i,],add=T,type="l")
}
```



You can see that in the first years the differentiation decreases sharply, because we started the simulation with the genotype frequencies distributed at random among demes and gene flow through dispersal can quickly homogeneize the allele frequencies. After a few years, the differentiation starts to increase slowly. This is the effect of genetic drift and partial isolation of some demes.

This example shows that the main steps for using MetaPopGen are: defining the input data, performing the simulation and analyzing the results.

4 Life cycle

The modelled life cycle consists of four consecutive phases: survival, reproduction, dispersal and recruitment (Fig. 1).

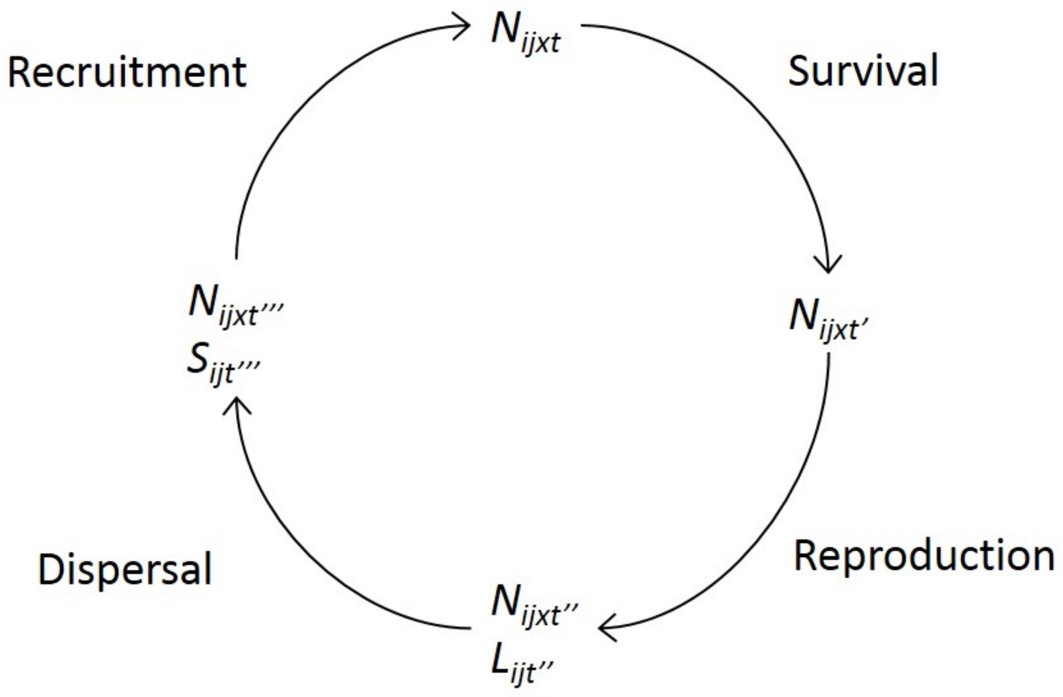


Figure 1: **Life cycle used in MetaPopGen.** *t* indicates time before survival; *t'* indicates time after survival and before reproduction; *t"* indicates time after reproduction and before dispersal; *t"'* indicates time after dispersal and before recruitment. N_{ijxt}: number of individuals of genotype *j* in deme *i* of age *x* at time *t*; $L_{ijt''}$: number of newborns of genotype *j* in deme *i* at time *t"*; $S_{ijt'''}$: number of juveniles of genotype *j* in deme *i* at time *t"'*.

The MetaPopGen code is based on equations regulating the relationships between N_{ijxt}, $L_{ijxt''}$ and $S_{ijxt'''}$ between each of the four phases of each time-step, and between time-steps. As an example, imagine a population of 100 individuals with survival $\sigma = 0.5$, fecundity $\phi_F = 5$, dispersal probability $\delta_{ij} = 0.2$ and recruitment probability $\sigma_0 = 0.8$. There will be about 50 (i.e. 100 times σ) surviving individuals out of 100; these 50 individuals will produce about 250 newborns (i.e. 50 times ϕ_F); of these newborns, about 125 (i.e. 250 times δ_{ij}) will be dispersed from deme *j* to deme *i*; of these 125 juveniles, about 100 (i.e. 125 times σ_0) will recruit as one-year-old individuals in the deme.
Because of the order of events in the life-cycle, the survival probability always applies at the beginning of the time-step. In a species with only one age-class, all the individuals must die at the end of the time-step; this does **not** mean that the survival probability $\sigma = 0$, because survival takes place at the beginning of the time

step; survival regulates how many individuals will take part in reproduction. Similarly, in the case of species with multiple age-classes, the individuals in the last age-class must die at the end of the time step; still, the last age-class can have a survival probability different from 0.

5 Input data

This section describes which input data you need to perform a simulation with MetaPopGen. We will recreate the input data for the Mediterranean example described in section 3.
The input data are of three types:
1. Demographic and genetic parameters
2. The initial genetic composition of the population
3. Control parameters, like the format of the input data and of the results.

5.1 Demographic and genetic parameters

The first variables to define are the number of **demes** `n` and the maximum **age** `z`. This will depend on the biology of your species and its geographical distribution. For the Mediterranean example, we set

```
n <- 115
z <- 5
```

Secondly, you should decide the number of simulated time steps `T_max`, i.e. the **length of the simulation**. This depends on the question you want to answer. If you are interested in studying genetic patterns at equilibrium, then you should make sure that `T_max` is sufficiently large for the population to reach an equilibrium. For the Mediterranean example, we set

```
T_max <- 200
```

Thirdly, you need to define the maximum number of **alleles** `l` and the maximum number of **genotypes** `m`. In MetaPopGen, the number of possible alleles cannot vary. This means that mutation cannot create new alleles: it can only change the frequencies of the predefined alleles. Nonetheless, if you are interested in the effect of new mutations, you can define a high number of possible alleles at the beginning of the simulation and set the frequencies of most of them at zero: this will treat those alleles as new mutations when they arise. For the Mediterranean example, we use only 2 alleles (say A1 and A2):

```
l <- 2
```

Therefore, the number of possible genotypes is three (A1A1, A1A2 and A2A2). In general, the number of genotypes can be calculated from the number of alleles:

```
m <- l*(l + 1)/2
```

These parameters (number of demes `n`, maximum age `z`, length of simulation `T_max` and number of genotypes `m`) are important because survival, fecundity and carrying capacity can vary according to the deme, age and genotype of the individuals and can be time-dependent.
To define survival probabilities, we first define a multidimensional array. Multidimensional arrays are ideal to store the value of a variables as a function of multiple dimensions. Here, the first dimension represents the genotype, the second dimension represents the deme, the third dimension represents the age and the fourth dimension represent the year. Because we are dealing with a species with separate sexes, we must define separate variables for male and females:

```r
sigma_M <- array(NA,dim=c(m,n,z,T_max))
dimnames(sigma_M) <- list(genotype=c("A1A1","A1A2","A2A2"),deme=c(1:n),
                          age=c(1:z),time=c(1:T_max))
sigma_F <- array(NA,dim=c(m,n,z,T_max))
dimnames(sigma_F) <- list(genotype=c("A1A1","A1A2","A2A2"),deme=c(1:n),
                          age=c(1:z),time=c(1:T_max))
```

Note that we have also assigned names to the dimensions (genotype, deme, age and time). Assigning dimension names is not necessary for running the simulations, but is helpful to interpret the variables. Note the order of the genotypes: we will return on the genotype order later on.
For both male and females, here we assign a **survival** probability of 0.8 to all genotypes, demes, age-classes and years:

```r
sigma_M[,,,] <- 0.8
sigma_F[,,,] <- 0.8
```

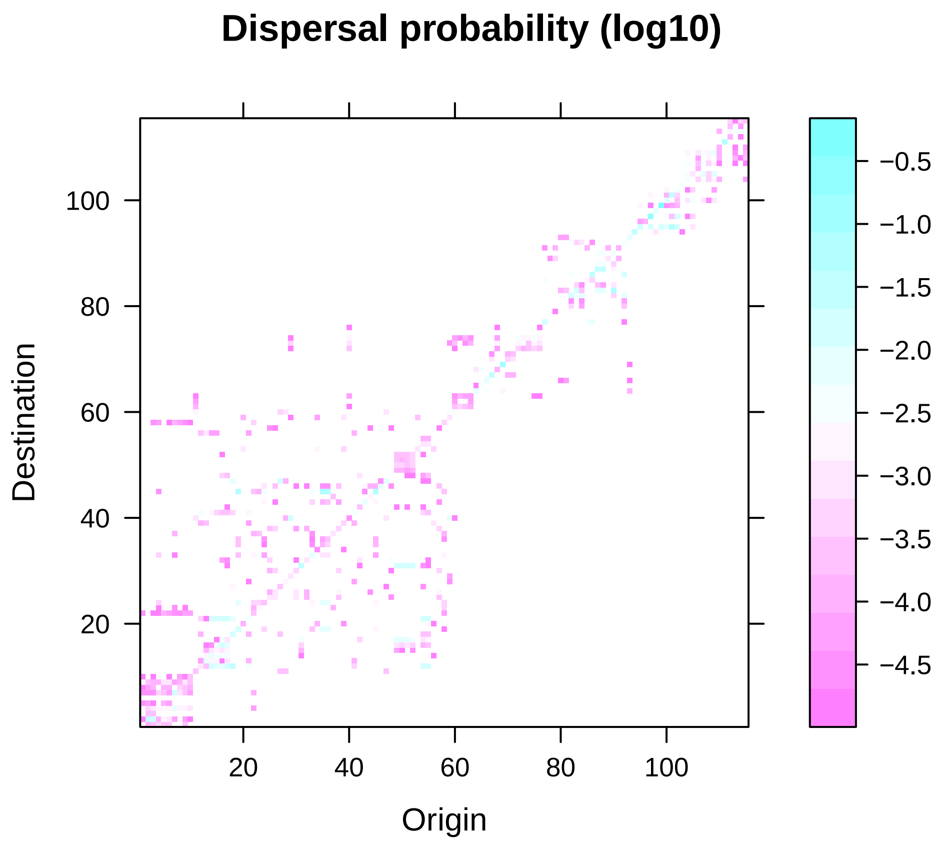
We proceed in the same way to define **fecundity**, but in this case we assign zero fecundity to the first two age-classes. The values of fecundities are representative of fish fecundities: a single female adult fish can produce hundred of thousands of eggs, here we choose 200000 eggs per capita. Male fecundity can be set to a large value, larger than the number of eggs, here 1 million.

```r
phi_M <- array(0,dim=c(m,n,z,T_max))
dimnames(phi_M) <- list(genotype=c("A1A1","A1A2","A2A2"), deme=c(1:n),
                        age=c(1:z), time=c(1:T_max))
phi_M[,,3:5,] <- 1000000

phi_F <- array(0,dim=c(m,n,z,T_max))
dimnames(phi_F) <- list(genotype=c("A1A1","A1A2","A2A2"), deme=c(1:n),
                        age=c(1:z), time=c(1:T_max))
phi_F[,,3:5,] <- 200000
```

Dispersal probabilities were estimated with a biophysical model of larval dispersal between the 115 demes. The variable `dispersal` is a 115 per 115 matrix, where elements *(i,j)* represent the probability of dispersal from deme j to deme i (and not from deme i to deme j!). The elements on the diagonal, *(i,i)*, are the probabilities that a larva settle in its deme of origin (local retention). To visualize the probability, a heatmap (or levelplot) is useful:

```r
levelplot(log10(t(dispersal)), xlab="Origin", ylab="Destination",
          main="Dispersal probability (log10)")
```



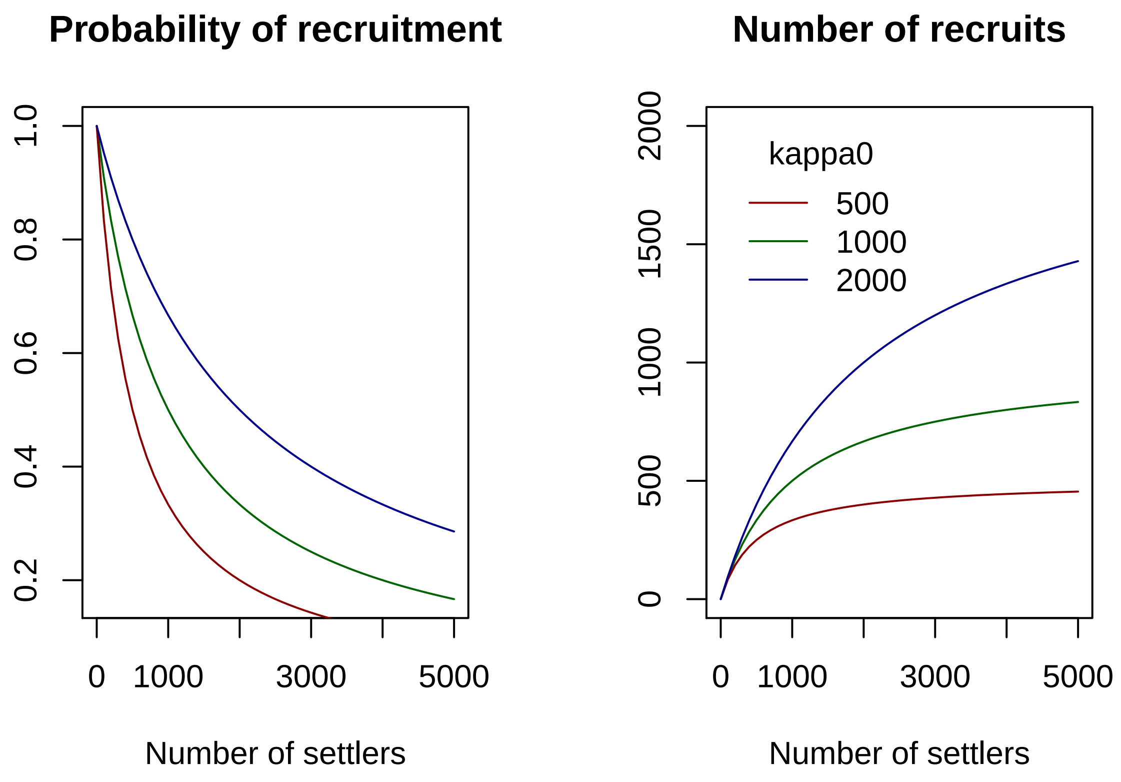
In MetaPopGen, the dispersal probabilities must be the same for male and females, and for all the genotypes, but they can be time-dependent. Therefore, we need to define a multidimensional array to define dispersal probabilities between demes for each time-step. Here, we use the same dispersal matrix for all years:

```
delta <- array(dispersal,dim=c(n,n,T_max))
dimnames(delta) <- list(destinationDeme=c(1:n),originDeme=c(1:n),time=c(1:T_max))
```

Settlers are all the individuals that settle in a deme after dispersal. **Recruits** are settlers that successfully recruit in the deme after density-dependent mortality. MetaPopGen implements two forms of density-dependent settler survival probability (i.e., the recruitment probability):
1. The survival probability of settlers depends inversely on the number of settlers themselves, mimicking competition among settlers, irrespective of the number of older individuals. The function used to calculate recruitment probability is:

$$1/(1 + S_{it}/k_{0it})$$

where S_{it} is the number of settlers and k_{0it} is the maximum number of settlers that can recruit in the deme (a sort of carrying capacity for the first age-class). The parameter `kappa0` can be deme- and time-dependent. The following figures illustrate the recruitment probabilities and the number of recruits for three values of `kappa0`.



2. The survival probability of settlers depends on the number of older individuals already present in the deme, mimicking competition among individuals for space. The maximum number of individuals allowed in the deme is `kappa0` (which can be deme- and time-dependent). If the number of individuals N_{tot} already present in the deme is larger than k_{0it}, settler survival will be set to zero. Otherwise, settler survival will be equal to

$$(k_{0it} - N_{tot})/S_{it}$$

Here, we use first model (competition among settlers) and we set `kappa0` to the same value (1000 recruits) for all demes and time-steps:

```
kappa0 <- array(1000,c(n,T_max),dimnames=list(deme=c(1:n),time=c(1:T_max)))
```

Mutation probabilities are stored in a matrix whose elements [i,j] define the probability that the allele j mutate into the allele j. We use a mutation probability of 1 out of one million for both mutations (A1 to A2 and A2 to A1)

```
mu <- array(1e-6,dim=c(l,l),dimnames=list(derivatedAllele=c("A1","A2"),
                                          ancestralAllele=c("A1","A2")))
mu[1,1] <- 1 - mu[2,1]
mu[2,2] <- 1 - mu[1,2]
```

5.2 The initial genetic composition of the population

The number of individuals of each genotype, for all demes and ages at time-step 1 must be defined. One common choice is to set the genotypic composition at random. We first define two variables, one for females and the other one for males, with the correct dimensions (genotype, deme and age-class)

```
N1_M <- array(NA,dim=c(m,n,z))
dimnames(N1_M) <- list(genotype=c("A1A1","A1A2","A2A2"),deme=c(1:n),age=c(1:z))
N1_F <- array(NA,dim=c(m,n,z))
dimnames(N1_F) <- list(genotype=c("A1A1","A1A2","A2A2"),deme=c(1:n),age=c(1:z))
```

Then we sample a random number of individuals for each genotype in each deme and age-class.

```
for (deme in 1 : n){
  for (age in 1 : z) {
    for (genotype in 1 : m) {
      N1_M[genotype,deme,age] <- round(runif(1)*300)
      N1_F[genotype,deme,age] <- round(runif(1)*300)
    }
  }
}
```

`runif()` is a random number generator that returns one number comprised between 0 and 1. The effect of scaling the random number generator by 300 leads to sample an average of 150 individuals. The rule of thumb for sampling x individuals in a random manner is to scale the random number generator by $2x$. Considering that there are `m = 3` genotypes, it means that we will have about 450 individuals per age class. This is in the same order of magnitude as the maximum number of recruits `kappa0` (1000).
Have a look at the genetic composition of the first deme. What does it mean?

```
N1_M[,1,]
```

```
##         age
## genotype   1   2   3   4   5
##     A1A1  13  16 269 263 154
##     A1A2 163   6  56 110 129
##     A2A2  85 242 297 230 125
```

It means that the genetic composition of individuals of age 1 is 13 A1A1, 163 A1A2 and 85 A2A2 (your numbers will likely be different from the ones of this example).

5.3 Control parameters

There is a number of parameters that control the simulation and can be passed to the function `sim.metapopgen.dioecious` as arguments:
`input.type` defines whether the input data are giving as arrays ("array") or as a data frame ("data.frame"). In the example above, we have used the array option. See below for an example using the data.frame option.
`save.res` tells the function whether the results of the simulations will be saved as files on disk (TRUE) or stored in a variable directly returned to the user (FALSE).
`save.res.T` If `save.res=TRUE`, this argument is used to define the time steps for which the state of the population will be saved on disk. It is used to save disk space in the case of long simulations with many demes or genotypes. It defaults to `seq(1,T_max)`, that is, the results are saved every time step.
`verbose` If TRUE, the function will print much information, mainly used for debugging purposes. Default to FALSE.

5.4 Input parameters as data.frame

Sometimes it is easier to build a table with the demographic parameters and the initial composition of the population. The following example shows the parameters for a dioecious populations with four demes, six age-classes, two alleles and five years.

```
rm(list=ls())
data(FourPopDataFrameDioecious)
ls()
```

```
## [1] "delta"            "demographic.data" "kappa0"
## [4] "mu"               "T_max"
```

```
head(demographic.data)
```

```
##   Genotype Deme Age Male_survival Female_survival Female_fecundity
## 1        1    1   1           0.4             0.4                0
## 2        2    1   1           0.4             0.4                0
## 3        3    1   1           0.4             0.4                0
## 4        1    2   1           0.4             0.4                0
## 5        2    2   1           0.4             0.4                0
## 6        3    2   1           0.4             0.4                0
##   Male_fecundity N1_M N1_F Time
## 1              0 1000 1000    1
## 2              0 1000 1000    1
## 3              0 1000 1000    1
## 4              0 1000 1000    1
## 5              0 1000 1000    1
## 6              0 1000 1000    1
```

The data.frame contains survival probabilities, fecundities and the initial genotypic compositions of the population. To launch a simulation using a data.frame input, you simply have to specify input.type="data.frame" and pass the data.frame as an argument to the function. The other demographic and genetic parameters (`delta`, `kappa0`, `mu` and `T_max`) are not included in the data.frame and must be passed separately.

```
## (Not run)
sim.metapopgen.dioecious(input.type="data.frame",
                         demographic.data=demographic.data, mu=mu,
                         delta=delta, kappa0=kappa0, T_max=T_max,
                         save.res=T)
```

5.5 More than two alleles and genotype order

The first dimension of many MetaPopGen objects (e.g. `sigma_M`, `phi_F`, `N_M`, ...) is the genotype. With two alleles (A1 and A2), the three genotypes are stored in this order: A1A1, A1A2 and A2A2. If there are three alleles (A1, A2 and A3), the order will be: A1A1, A1A2, A1A3, A2A2, A2A3, A3A3. The rules are: take the first allele, store the homozygote first, then all the heterozygotes including the first allele, then step to the subsequent alleles. To know the place of the $i^{th}j^{th}$ genotype, you can use the function "genotype.index", which gives a matrix containing the position of the genotype as a function of the number of alleles. For example, if you want to know the position of the A2A3 genotype when there are three alleles:

```
y <- genotype.index(3)
y
```

```
##    A1 A2 A3
## A1  1  2  3
## A2  2  4  5
## A3  3  5  6
```

```
y[2,3]
```

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

The A2A3 genotype is in the fifth position. Obviously, the A2A3 and A3A2 genotype are the same:

```
y[3,2]
```

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

For most applications you will not have to worry about the order of the genotypes.

6 Analyzing the results

In this section, we explore the results of the Mediterranean simulation performed in section 3 (A simple example to get started). If you have completed section 3, the results of the five replicate simulations are stored in a folder named "Simulation.Med". In each subfolder (one for each replicate and named with the date and the time of the beginning of that simulation), there are the simulation results: one file for each year of simulation. Consider only one replicate for the moment. Let's have a look at the N1.RData and the N200.RData files, which contains the initial and the final genotypic composition of the simulation.

```
name.dir <- paste0(getwd(),"/Simulation.Med/2016-juil.-22-11.48.29")
# (Change the name of the folder according to the name of your simulation)

load(paste0(name.dir,"/N1.RData"))
N1_M <- N_M; N1_F <- N_F
load(paste0(name.dir,"/N200.RData"))
N200_M <- N_M; N200_F <- N_F
```

The format of these variables is the same as the ones defining the initial genotypic composition of the population: multidimensional arrays whose dimensions are, in order: genotype, deme and age-class. Let's assign dimension names to the variables:

```
dimnames(N1_M) <- list(genotype=c("A1A1","A1A2","A2A2"),
                       deme=c(1:115),age=c(1:5))
dimnames(N1_F) <- dimnames(N1_M)
dimnames(N200_M) <- dimnames(N1_M)
dimnames(N200_F) <- dimnames(N1_M)
```

Type the name of the variables to see the genotypic composition. For example, print the genotypic composition of the males of the first age class in the first deme at time 1 :

```
N1_M[,1,1]
```

```
## A1A1 A1A2 A2A2
##   18  299  277
```

and at time 200 :

```
N200_M[,1,1]
```

```
## A1A1 A1A2 A2A2
##  124  220  112
```

At the beginning of the simulation, in the first age-class of deme 1 there are 18 male individuals of genotype A1A1, 299 of genotype A1A2 and 277 of genotype A2A2. At the end of the simulations, these numbers have changed to 124, 220 and 112, respectively.
The number of genotypes is what we need to calculate most genetic population parameters.

6.1 Allele frequencies

The allele frequencies of a deme or a group of individuals can be calculated with the function “freq.all”. The function returns a vector containing the frequencies of all the alleles. In our case, the first element of the vector is the frequency of the A1 allele and the second element is the frequency of the A2 allele. Look at the examples below, which give the allele frequencies of one-year-old males in deme 1 at the beginning and at the end of the simulation.

```
# Beginning of the simulation
freq.all(N1_M[,1,1])
```

```
## [1] 0.2819865 0.7180135
```

```
# End of the simulation
freq.all(N200_M[,1,1])
```

```
## [1] 0.5131579 0.4868421
```

The only argument of the function is a vector of genotype numbers: a vector is a one-dimensional array. In the example above, we chose only one deme and one age-class, and only one sex (males). If we want the allele frequencies of the males of an entire deme, we have to sum the N array accordingly:

```
N1_M[,1,]
```

```
##          age
## genotype   1   2   3   4   5
##     A1A1  18 185 170 191 270
##     A1A2 299 273  43 147 289
##     A2A2 277 164 126 126 150
```

```r
x <- apply(N1_M[,1,],1,sum)
x
```

```
## A1A1 A1A2 A2A2
##  834 1051  843
```

```r
freq.all(x)
```

```
## [1] 0.4983504 0.5016496
```

`N1_M[,1,]` is a 2-D array containing the first (genotype) and the third (age-class) dimensions of the `N1_M` array (which has three dimensions: genotype, deme and age-class). The `apply` function is used to apply the `sum` function to the elements of the three rows of `N1_M[,1,]`. The second argument of `apply` is equal to 1 because rows are the first dimension of `N1_M[,1,]`. If we wanted to apply the `sum` function to the columns of `N1_M[,1,]`, the second argument of `apply` would be 2. In this case, the same result may have been obtained with `rowSums(N1_M[,1,])`. The result of the `apply` function is vector `x`, which contains the number of individuals for each genotype in the first deme, over all age-classes.
To calculate the frequencies over males and females:

```r
freq.all(apply(N1_M[,1,],1,sum) + apply(N1_F[,1,],1,sum))
```

```
## [1] 0.5129792 0.4870208
```

We can also calculate the frequency of the the various age-classes, for example to compare the allele frequencies of the first and the second age-class of deme 1:

```r
# The first age-class
freq.all(N1_M[,1,1] + N1_F[,1,1])
```

```
## [1] 0.4489974 0.5510026
```

```r
# The second age-class
freq.all(N1_M[,1,2] + N1_F[,1,2])
```

```
## [1] 0.5023518 0.4976482
```

If we want to compare the frequencies of the first and second age class over the entire population:

```r
# The first age-class
freq.all(apply(N1_M[,,1],1,sum)+apply(N1_F[,,1],1,sum))
```

```
## [1] 0.4973667 0.5026333
```

```r
# The second age-class
freq.all(apply(N1_M[,,2],1,sum)+apply(N1_F[,,2],1,sum))
```

```
## [1] 0.5101699 0.4898301
```

Finally, to calculate the allele frequencies of the entire population:

```r
freq.all(apply(N1_M,1,sum)+apply(N1_F,1,sum))
```

```
## [1] 0.5058992 0.4941008
```

In all these examples, the allele frequencies are always close to 0.5 and do not differ much between demes or age-classes, because the initial genotypic composition of the population was determined by choosing the number of genotypes at random and with equal probabilities.

6.2 Genetic differentiation: F_{ST}

Differences in allele frequencies can be used to measure the genetic differentiation between groups of individuals. The F_{ST} is a common index to measure genetic differentiation. There are various estimators of F_{ST} for data collected from natural populations, for example the θ of Weir and Cockerham (1984). The estimators of F_{ST} include correction for finite sample size and finite number of sampled demes. In the case of simulated data, such corrections are not needed, because we assume that all individuals of all demes are represented in the output of the simulations. Thus, F_{ST} can be calculated as ratio of heterozygosities. The function `fst` calculate the F_{ST} between groups of individuals given a two-dimensional array containing the number of individuals of each genotype (first dimension: rows) in each group (second dimension: columns). There must be at least two groups to calculate F_{ST} with the function `fst`, but there is no limit to the maximum number of groups.
In the Mediterranean example, we now calculate the genetic differentiation between deme 1 and deme 50 at the end of the simulation. The first thing to do is to calculate the vector of genotype frequencies for the groups; then, the genotype frequencies are passed to the function `fst` as a 2-D array.

```r
N1 <- apply(N200_M[,1,],1,sum) + apply(N200_F[,1,],1,sum)
N50 <- apply(N200_M[,50,],1,sum) + apply(N200_F[,50,],1,sum)
N <- array(c(N1,N50),dim=c(3,2),
           dimnames=list(genotype=c("A1A1","A1A2","A2A2"),
                         group=c("deme1","deme50")))
fst(N)
```

```
## [1] 0.00469197
```

We can see that the F_{ST} is quite small.

Now we calculate the genetic differentiation among all demes. We have to sum over age-classes:

```r
N <- apply(N200_M,c(1,2),sum) + apply(N200_F,c(1,2),sum)
```

Note the argument `c(1,2)` passed to the apply function, which means that the result of apply will have two dimensions, corresponding to the first and the second dimension of `N200_M`: genotype and deme. The same result may be obtained with

```r
NN <- apply(N200_M,c("genotype","deme"),sum) +
  apply(N200_F,c("genotype","deme"),sum)
all(N == NN)
```

```
## [1] TRUE
```

Finally, calculate the F_{ST}:

```
fst(N)
```

```
## [1] 0.007295752
```

7 Exercises

7.1 Defining input data

Define the input data for a plant population with the following features:
1. Monoecious species with three age-classes and four demes; two alleles; time of simulation: 50 years.
2. The survival probability is 0.5 for the first age-class and 0.8 for the other two age-classes
3. Female fecundity is equal to 30 embryo sacs (i.e. female gametes) per capita; male fecundity is equal to 1000 pollen grains (i.e. male gametes) per capita for all age-classes
4. The carrying capacity of local demes is 15 adults (settler survival probability depends on the total number of individuals in the deme) and is constant in time.
5. Mutation rate: 10^{-6}
6. Dispersal probabilities as in Figure 2.
7. The initial number of individuals is about 5 per age-class per deme, equally distributed among genotypes.

7.2 2. Performing the simulation and analyzing the results

Using the data prepared in the previous exercise, perform a simulation and save the results on disk. Calculate the allele frequencies of the first allele for each deme at each time step. Plot the allele frequencies as a function of time, as in the example graph below.

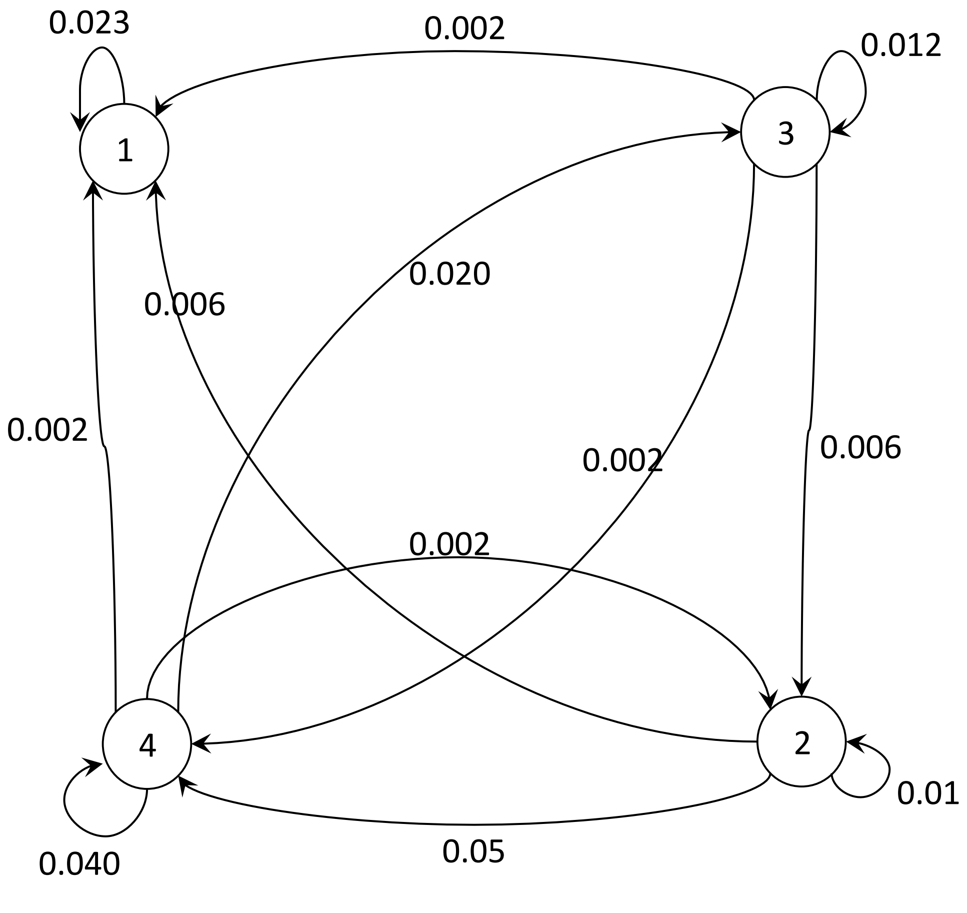


Figure 2: Exercise 1: dispersal probabilities between demes

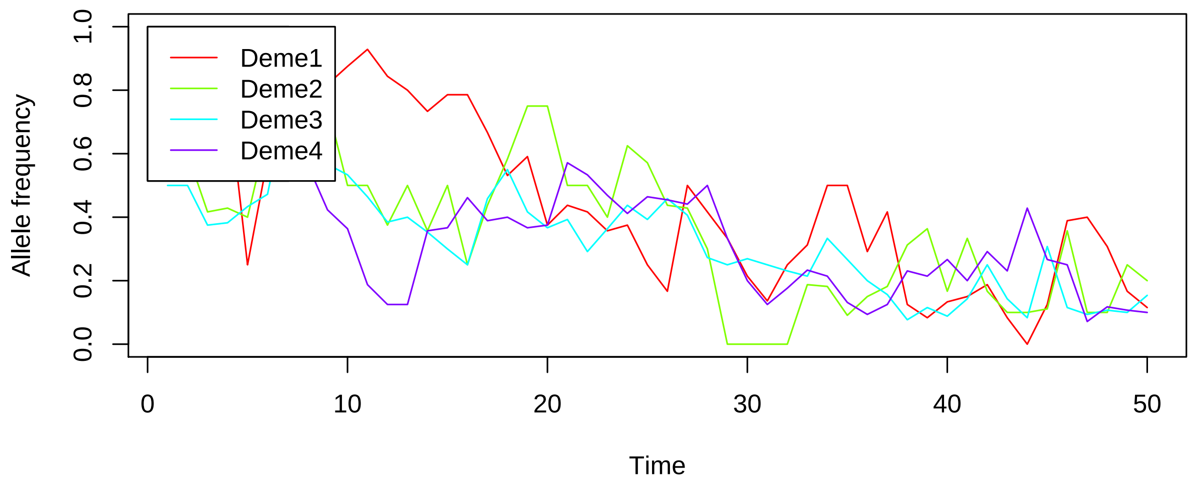


Figure 3: Exercise 2: Frequency of the first allele as a function of time in the four demes