Quick tutorial on how to use RQuantiNemo

Introduction

RQuantiNemo is a small R package which allows to bring all the workflow of quantinemo simulation into R. It is written to act at three different stages: * Write an input file, and accessory files (like the quanti_allelic file) from a set of parameters and dataframe * Run a simulation * retrieve the result

It is based on a S4 object, called a simulation. A simulation can be created with a set of parameter for quantiNemo, launch, and resutl can be loaded from it. In this document, we will show three different example. First a toy one doing nothing, then a simple one with just a more advance demographic scenario, then a third one with more input and output file in a case of quantitive genetics.
### Launching a minimal simulation To Launch a minimal simulation, we just have to create one object simulation, run the simulation using the command run(). Here, we just add one parameter to have an output file to check that the simulation was actually performed.

```
devtools::load_all(pkg="../../RQuantiNemo")
```

```
## Loading RQuantiNemo
```

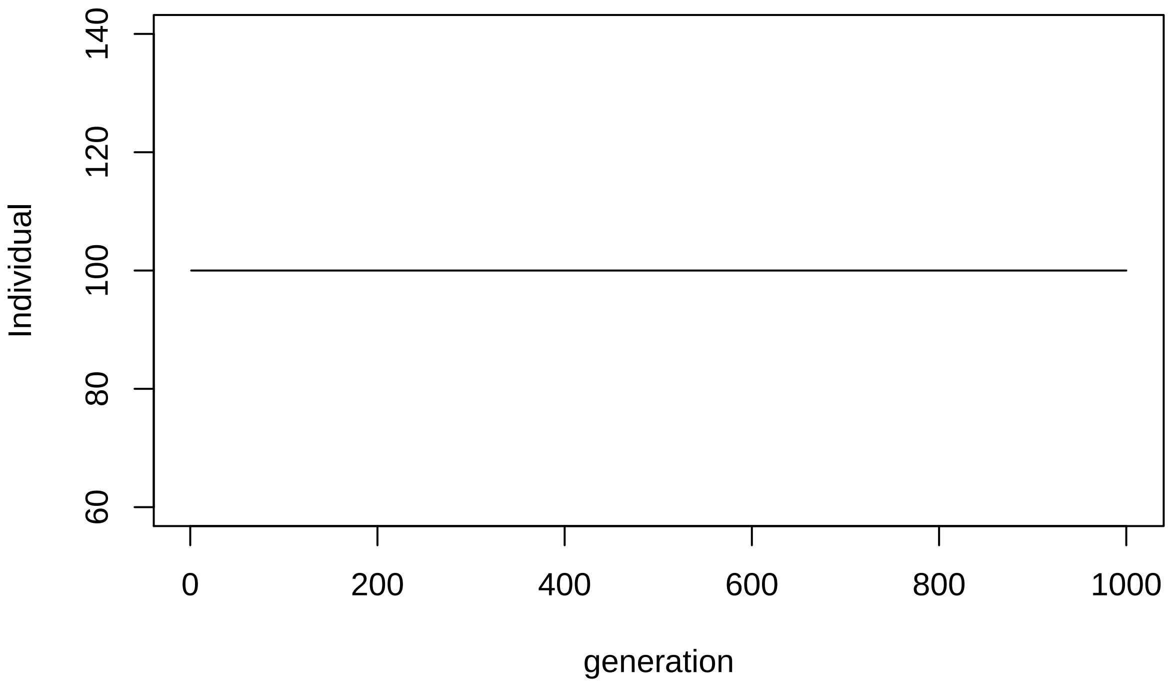
```
my_simulation <- new("simulation") # some default parameter will be set for us
my_simulation <- setParameter(my_simulation,"stat","{adlt.nbInd}")
run(my_simulation)
```

```
## printing into :./my_simulation.ini

## excuting /Users/frederic/Desktop/RQuantiNemo/quantiNemo2 ./my_simulation.ini

## [1] 0
```

```
statistics <-loadStat(my_simulation)
plot(statistics$adlt.nbInd, t="l",xlab = "generation", ylab = "Individual")
```



As we can see, the output of quantiNemo is visible from the terminal. The population size was set by default

to 100, and the number of generation to 1000. If we check in our favorite file navigator, we will see that the simulation ran in the current directory, outputting file here. As we will see it, this behaviours might be changed.
Then, the function loadStat read the statistics from the simulation, and return a dataframe of them. As we can see from the plot, we simulate one patch with 100 individuals for 1000 generations

A more advance demographic example

We want to explore now with a bit more detail the exact possibility of quantiNemo in term of file handling and inputing more advance parameters. We first define a list containing the parameters that we want, and create a new simulation from this list of parameter. We also specify a new "name" and "working directory" in order not to remove our prevous work

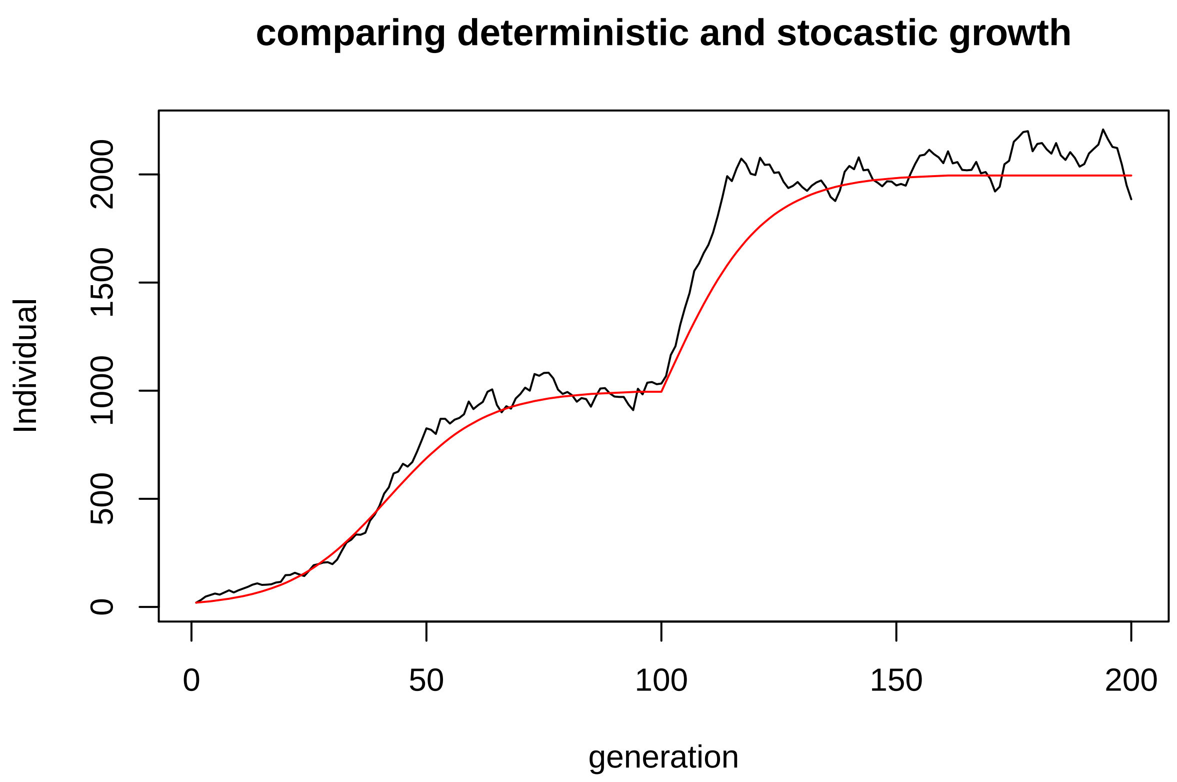
```
my_parameters = list("generations" = 200,
                  "patch_capacity" = "(1 1000, 100 2000)",
                  #"advanced" parameters are specified through character
                  "patch_ini_size" = 20,
                  "mating_nb_offspring_model" = 9, # A logistic-stochastic distribution
                  "growth_rate" = 0.1,
                  "stat" = "{adlt.nbInd}"
)
# We specify the parameters, and also the name and the directory
# where the simulation will be performed
# The input file will be demography/stochastic.ini.
# The output file will be in demography/stochastic/
my_simulation <- new("simulation", parameters = my_parameters,
                     sim.name = "stochastic", sim.dir = "demography/")
run(my_simulation, verbose = F) #we don't want to have the output of QN printed
```

```
## [1] 0
```

```
stat.stocastic <- loadStat(my_simulation)
# We now want to compare this to the same simulation but removing the stochastic component
my_simulation <- setParameter(my_simulation,"mating_nb_offspring_model", 8)
#We want now the simulation to be outputed in demography/deterministic
my_simulation@sim.name <- "deterministic" #Probably I should add a setter for that.
run(my_simulation, verbose = F) #we don't want to have the output of QN printed
```

```
## [1] 0
```

```
stat.deterministic <- loadStat(my_simulation)
#Plotting the result of both simulation
plot(stat.stocastic$adlt.nbInd, t="l",xlab = "generation", ylab = "Individual",
     main = "comparing deterministic and stocastic growth")
lines(stat.deterministic$adlt.nbInd, col = "red")
```



Note that the two 0 outputed in the console mean that both simulation run correctly.

Adding some quantitative trait

We wants now to investigate the effect of recombination on the fitness of individuals in a population with deleterious mutation occuring. To do so, we set a population with one chromosome of a certain length which can be loaded with deleterious mutation due to mutation. We want to investigate how the average number of deleterous loci vary as a function of the recombination proportion.

```
my_parameters = list(
                  "generations" = 300,
                  "replicates" = 20,
                  "patch_capacity" = 100,
                  "selection_pressure_definition"    = 1,       # at qt level
                  "mating_system" = 3,      # promiscuity
                  "selection_level" =   0,      # soft selection
                  "quanti_environmental_proportion" =   0,
                  "quanti_nb_trait" =       1,
                  "quanti_all" = 2,
                  "quanti_loci" =       50,
                  "quanti_genome"=          "{seq(0,1,50)}",
                  "quanti_locus_index"=     "{seq(1,50,50)}" ,
                  "quanti_mutation_rate"=       "0.001" ,
                  "quanti_selection_model"= 0,      # neutral selection
                  "stat"    =   "{
                              q.meanWfem_p
                                  q.meanWmal_p
                                  q.meanPfem_p
                                  q.meanPmal_p
                                q.varPfem_p
                                q.varPmal_p
                                  meanW_p
                            }",
                  "quanti_save_genotype"= 1
)

allele_distrib = data.frame("col_allele" = c(1, 2),
                            "col_allelic_value" = c(0, 1),
                            "col_mut_freq" = c(0, 1),
                            "col_ini_freq" =c(1,0))

geno_fitness = data.frame("col_allele1" = c(1, 1, 2),
                            "col_allele2" = c(1, 2, 2),
                            "col_fitness_factor" = c(1, 0.999,0.9))

print(allele_distrib)
```

```
##   col_allele col_allelic_value col_mut_freq col_ini_freq
## 1          1                 0            0            1
## 2          2                 1            1            0
```

```
print(geno_fitness)
```

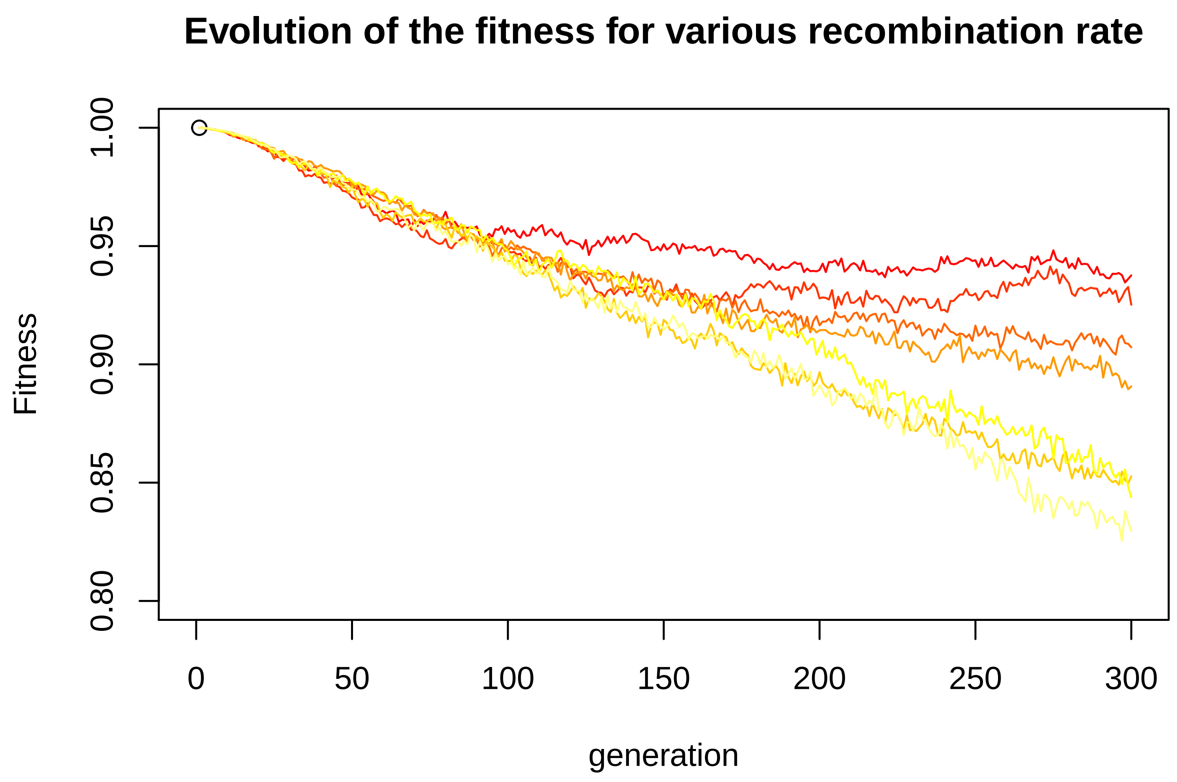
```
##   col_allele1 col_allele2 col_fitness_factor
## 1           1           1              1.000
## 2           1           2              0.999
## 3           2           2              0.900
```

```
my_simulation <- new("simulation", parameters = my_parameters, sim.dir = "genetic/")
my_simulation <-  addFile(my_simulation, "quanti_allelic_file",allele_distrib)
my_simulation <-  addFile(my_simulation, "quanti_dominance_file",geno_fitness)
```

Now that the input file is ready, we will do a loop and launch several simulation with different parameter "recombination_factor", and display the result of the different simulation.
We see that the package allows to do similar thing as the "batch mode" of quantinemo, i.e. launching easly several simulations. The advantage of QN is that it's a bit more straightforwards (just list the different value of the parameter). The advantage of the package is that it's much more flexible in term of changing the parameter and deciding the name of the different output file

```
chromosome.lengthes = 100^(seq(1,-0.5,-0.25)) # The length of the chromosome.
#Notice that this would be difficult to do with Macro.
statistics = list()
for (chromosome.length in chromosome.lengthes){
  #For normal parameter we should just be able to put directly the value
  #The recombination factor expect a matrix so we have to convert the value to a character
  my_simulation <- setParameter(my_simulation, "recombination_factor",paste("{" ,as.character(chromosome
  my_simulation@sim.name = as.character(chromosome.length) #We set a
  run(my_simulation, verbose = F)
  statistics[[length(statistics)+1]] <- loadStat(my_simulation)
}
```

```
plot(1, axes=T,ylim = c(0.8,1),xlim = c(0,300), xlab="generation",
     ylab="Fitness", main = "Evolution of the fitness for various recombination rate")
palette(heat.colors(length(statistics)))
for (i in 1:length(chromosome.lengthes)){
  lines(statistics[[i]]$meanW, col = i)
}
```



Finally, we can also load the fstat file for any generation and replicate (if it was saved)

```
geno <- loadGeno(my_simulation, replicate = 1, generation = 99)
geno2 <- as.matrix(geno[,2:51])
table(geno2) #How much deleterious mutation we have in the population
```

```
## geno2
##   11   12   21   22
## 4507  218  238   37
```