Evolution of demographic parameters and population stability

Computational biology project by Bianca BOI and Anne STAFF

```
set.seed(42)
library(tidyverse)
```

Our model

We initialise the model with the following values:

```
# initial values
K <- 950 # equilibrium density, we interpreted
# "relaitevly small" as being close to the initial total density
lambda <- 1.5 # growth rate, also "relatively small"
N_init <- 1000 # initial total density
c <- -2 # complexity, less than -1
mu <- 0.01 # mutation rate, given
# we will define later r as the relative density à savoir N/mean(K)

# population initiale
population <- data.frame(K=rep(K, N_init),
                         lambda=rep(lambda, N_init),
                         c=rep(c, N_init))
```

We define a function that calculates the fitness value from the given traits:

```
# calculate fitnesses
calc_fitness <- function(N, K, lambda, c) {
  # make sure we're not accidentally dividing by 0
  if (any(lambda == 1)) {
    lambda <- 1.0000000001
  }
  res <- lambda / (1 + (lambda - 1)*((N/K)**((1-c)*lambda/(lambda-1))))
  return(res)
}
# test
# with(population[1,], calc_fitness(N_init, K, lambda, c))
```

We calculte the initial fitness values for our initial population:

```
# run
fitness <- with(population[1,], calc_fitness(N_init, K, lambda, c))
population$fitness <- rep(fitness, N_init)
```

We define a function that creates the offspring from one individual

```
# offspring per individual without mutation
offspring <- function(individual) {
  n_offspring <- rpois(1, individual[4])
  if (is.na(n_offspring)) {
    return(vector())
```

```
  } else {
    offspring <- data.frame(K=rep(individual[1], n_offspring),
                            lambda=rep(individual[2], n_offspring),
                            c=rep(individual[3], n_offspring),
                            fitness=rep(individual[4],
                                        n_offspring))
    return(offspring)
  }
}
# test
# offspring(population[1,])

# also, an optimised fitness-calculation
# for vectorising the operations is needed:
vector_fitness <- function(input, N) {
  lambda = input[2]
  if (any(lambda == 1)) {
    lambda <- 1.0000000001
  }
  res <- lambda / (1 + (lambda - 1)*
                     ((N/input[1])**((1-input[3])*lambda/(lambda-1))))
  return(res)
}
```

We define a function that computes the next generation for a given population, using the offspring function. At this point, mutations are not taken into account yet (wait for it).

```
# next generation
next_generation <- function(population){
  next_gen <- apply(population, 1, offspring)
  next_gen <- bind_rows(next_gen)
  next_gen$fitness <- apply(next_gen,
                            1,
                            vector_fitness,
                            N=nrow(next_gen))
  return(next_gen)
}
# test
# next_generation(population)
```

TADAAAAA mutation appears:

```
next_generation_with_mutation <- function(population){
  next_gen <- next_generation(population)
  n_mutations <- rpois(1, nrow(next_gen)*mu)
  mutants <- sample(nrow(next_gen), n_mutations, replace=FALSE)
  if(length(mutants) > 0) {
    for(i in 1:(ncol(next_gen)-1)) {
      next_gen[mutants,i] <- next_gen[mutants, i] + rnorm(1, 0, i*0.04)
    }
    next_gen[mutants,4] <- with(next_gen[mutants,],
                                calc_fitness(nrow(next_gen), K, lambda, c))
  }
  return(next_gen)
}
```

Just like before, we create the offspring for each individual of the population. Then, out of this newly created generation, we draw the number of mutants from a poisson distribution with a probability of 0.01, pick random mutants from the new generation and modify their traits accordingly.

Here, we define functions that run the model over a given number of generations:

```
# run over x generations and plot evolution of population size etc
run <- function(x, population) {
  generations <- population[1,]
  generations$N <- nrow(population)
  # without mutation
  for (i in 1:(x-1)) {
    population <- next_generation(population)
    population$N[1] <- nrow(population)
    generations <- rbind(generations, population[1,])
  }
  return(generations)
}
```

Same thing with mutations:

```
run_with_mutation <- function(x, population, verbose=FALSE) {
  if (verbose) {
    print(c("Generation", "Number of individuals"))
    print(c(1, nrow(population)))
  }
  # with mutation
  generations <- data.frame(N=nrow(population),
                                          K_means=mean(population$K),
                                          K_sd=sd(population$K),
                                          lambda_means=mean(population$lambda),
                                          lambda_sd=sd(population$lambda),
                                          c_means=mean(population$c),
                                          c_sd=sd(population$c),
                                          fitness_means=mean(population$fitness),
                                          fitness_sd=sd(population$fitness))
  for (i in 1:(x-1)) {
    population <- next_generation_with_mutation(population)
    if (verbose) {
      print(c(i+1, nrow(population)))
    }
    generations <- rbind(generations,
                         c(nrow(population),
                           mean(population$K),
                           sd(population$K),
                           mean(population$lambda),
                           sd(population$lambda),
                           mean(population$c),
                           sd(population$c),
                           mean(population$fitness),
                           sd(population$fitness)))
  }
  generations$r <- generations$N / generations$K_means
  return(generations)
}
```

Execute:

```
x <- 20
generations <- run(x, population)
generations_with_mutation <- run_with_mutation(x, population)
```

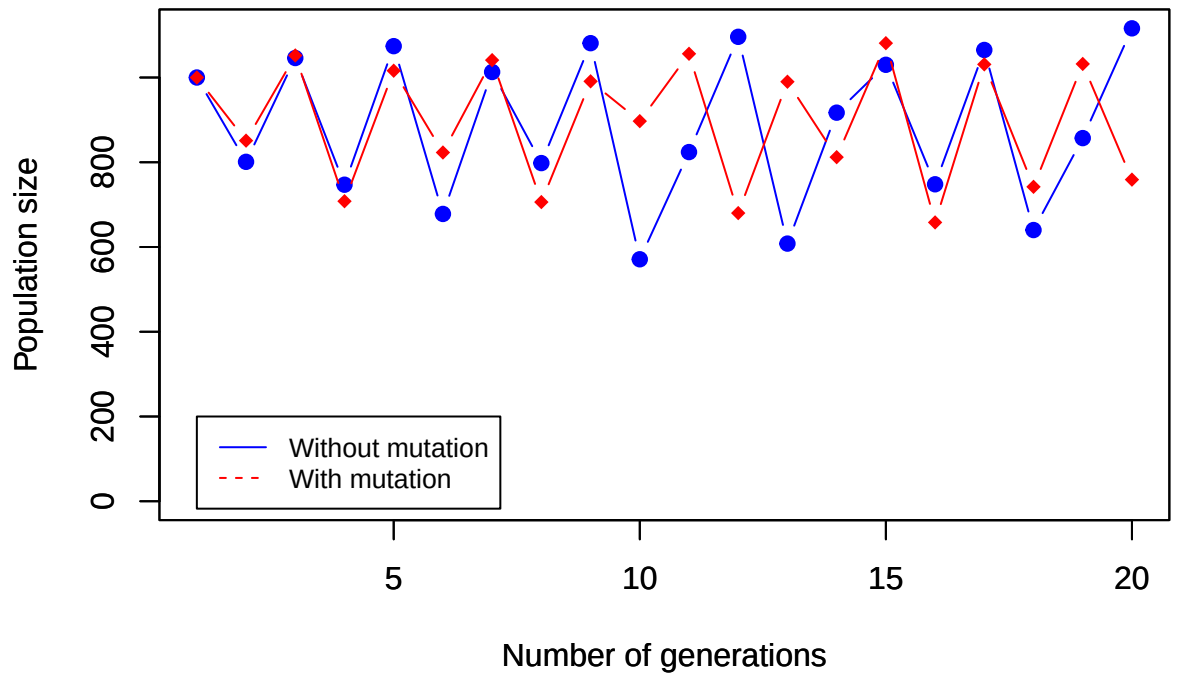
Now plot:

```
plot(1:x,
  generations$N,
  type = "b",
  pch = 19,
  col = "blue",
  xlim = c(1, x),
  ylim = c(0, max(
    c(generations$N, generations_with_mutation$N)
  )),
  xlab = "Number of generations",
  ylab = "Population size",
  main = "Evolution of population sizes over generations with and without mutations")

par(new = T)

plot(1:x,
  generations_with_mutation$N,
  type = "b",
  pch = 18,
  col = "red",
  xlim = c(1, x),
  ylim = c(0, max(
    c(generations$N, generations_with_mutation$N)
  )),
  xlab = "Number of generations",
  ylab = "Population size")

legend(1,
  200,
  legend = c("Without mutation", "With mutation"),
  col = c("blue", "red"),
  lty = 1:2,
  cex = 0.8)
```


Evolution of population sizes over generations with and without mutat
Population size
0
200
400
600
800
5
10
15
20
Number of generations
Without mutation
With mutation