Evolution of demographic parameters and population stability

```
set.seed(42)
```

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 d
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) {
  n_offspring <- rpois(1, individual$fitness)
  if (is.na(n_offspring)) {
    return(vector())
  } else {
    offspring <- data.frame(K=rep(individual$K, n_offspring), lambda=rep(individual$lambda, n_offspring
    return(offspring)
  }
}
```

```
# test
# offspring(population[1,], N_init)
```

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 <- data.frame(K=vector(), lambda=vector(), c=vector(), fitness=vector())
  for (i in 1:nrow(population)){
    o <- offspring(population[i,], nrow(population))
    next_gen <- rbind(next_gen, o)
  }
  return(next_gen)
}
# test
# next_generation(population)
```

TADAAAAA mutation appears:

```
next_generation_with_mutation <- function(population){
  next_gen <- data.frame(K=vector(), lambda=vector(), c=vector(), fitness=vector())
  for (i in 1:nrow(population)){
    o <- offspring(population[i,], nrow(population))
    next_gen <- rbind(next_gen, o)
  }
  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) {
  population_with_mutation <- population
  if (verbose) {
    print(c("Generation", "Number of individuals"))
    print(c(1, nrow(population_with_mutation)))
  }
  # with mutation
  generations_with_mutation <- data.frame(N=nrow(population_with_mutation), K_means=mean(population_wit
  for (i in 1:(x-1)) {
    population_with_mutation <- next_generation_with_mutation(population_with_mutation)
    if (verbose) {
      print(c(i+1, nrow(population_with_mutation)))
    }
    generations_with_mutation <- rbind(generations_with_mutation, c(nrow(population_with_mutation), mea
  }
  generations_with_mutation$r <- generations_with_mutation$N / generations_with_mutation$K_means
  return(generations_with_mutation)
}
```

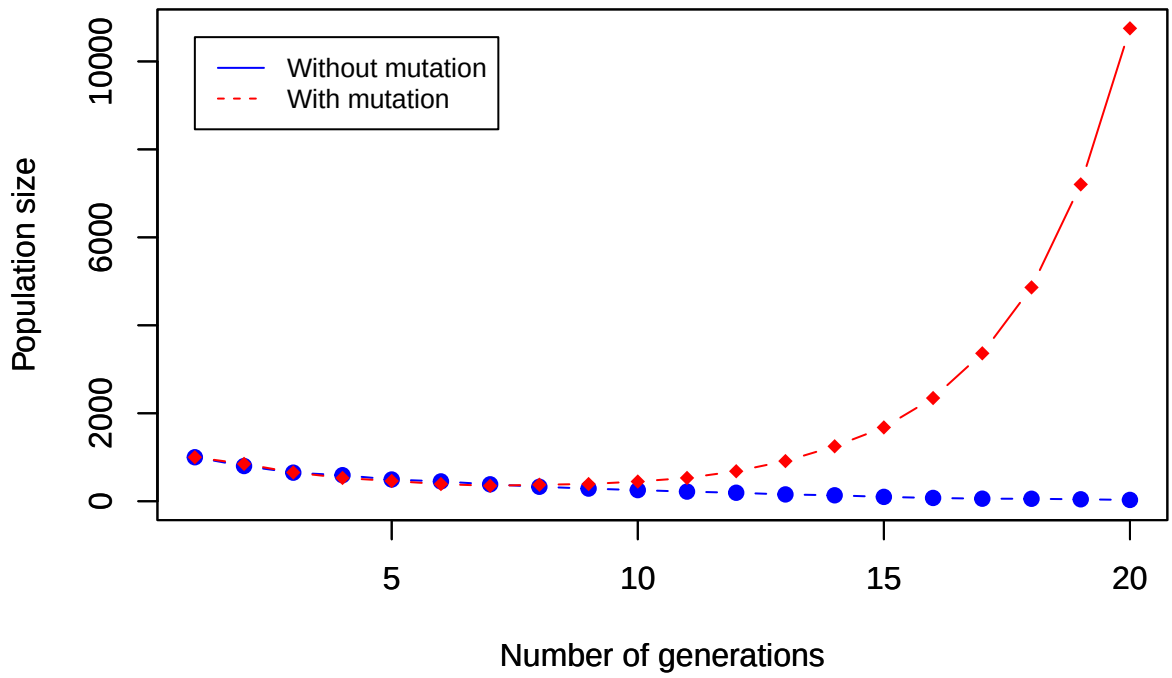
Execute:

```
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, gen
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(genera

legend(1, max(c(generations$N, generations_with_mutation$N))-200, legend=c("Without mutation", "With mu
```



We can see that with mutation, the population density increases exponentially, as in the paper, where it is represented with a logarithmic coordinate. Being unaware of the exact initial values used in the paper, we decided after some playing around, to go with an initial population of N= 1000 and K=950. The problem we encounter at this point, is that the calculation time explodes as it increases exponentially too, just as the population density. In fact, even with smaller initial population sizes we barely make it to 20 generations after 10 minutes of computation. All our attempts to go further than that failed since even for as little as 25 generations the number of individuals usually already exceeds 100k and the computation times get completely out of hand.

At the same time, the equilibrium density seems to barely vary across generations, unlike in the paper. Even at a total density around 40k individuals the equilibrium density seems to not surpass 950.2 at most (so barely a 0.2 difference compared to the starting value). So we're unsure if limiting this value will make any significant difference. Also, we're unsure as to which value we should limit it to.

Judging by the graphs in the paper, the initial equilibrium density seems to be about 5000 (10^3.7), and the number of generations they compute is more than 200k. With our current code we won't get anywhere close to these numbers.

Some optimisation of the code (avoiding explicit for-loops for example) could probably improve the performance. However, at the moment, this exceeds our R syntax expertise.

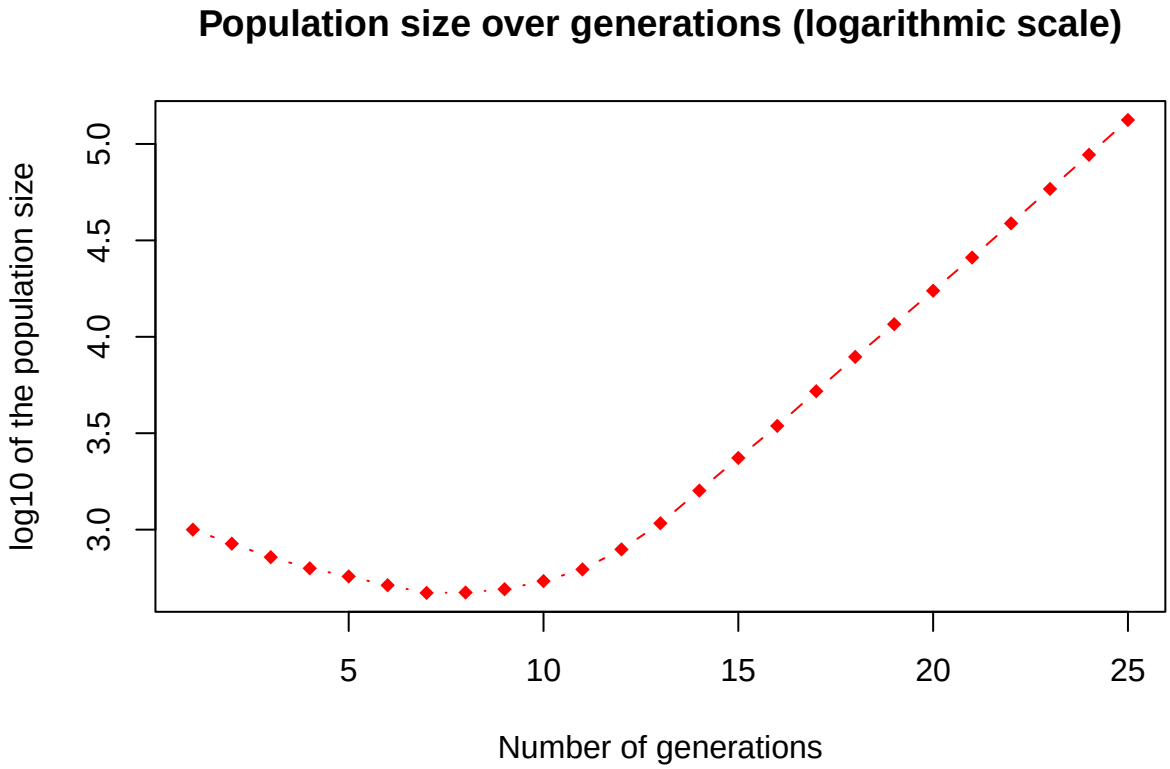
Some plots:

```
x <- 25
gen <- run_with_mutation(x, population, TRUE)
```

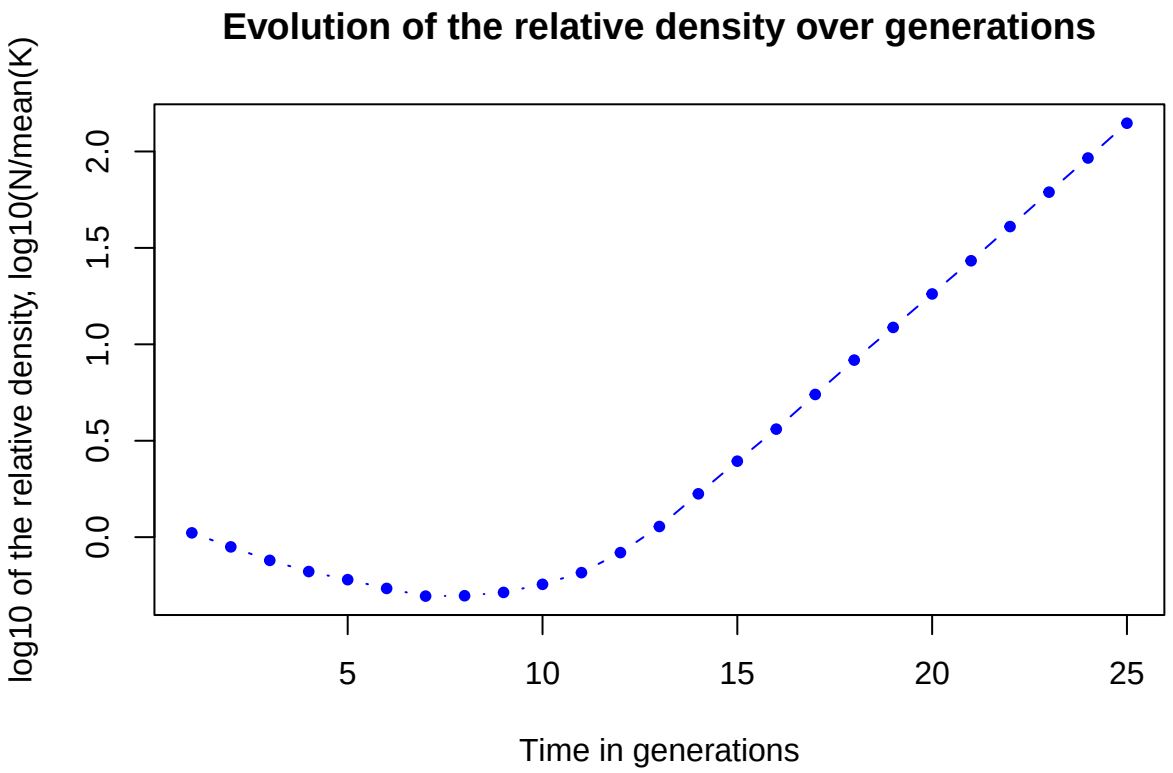
```
## [1] "Generation"           "Number of individuals"
## [1]    1 1000
```

```
## [1]   2 846
## [1]   3 720
## [1]   4 630
## [1]   5 572
## [1]   6 515
## [1]   7 470
## [1]   8 472
## [1]   9 491
## [1]  10 541
## [1]  11 622
## [1]  12 790
## [1]   13 1079
## [1]   14 1594
## [1]   15 2354
## [1]   16 3451
## [1]   17 5220
## [1]   18 7868
## [1]    19 11625
## [1]    20 17333
## [1]    21 25760
## [1]    22 38760
## [1]    23 58445
## [1]    24 87876
## [1]     25 133122
```

```
plot(1:x, log10(gen$N), type="b", pch=18, col="red", xlab="Number of generations", ylab="log10 of the p
```

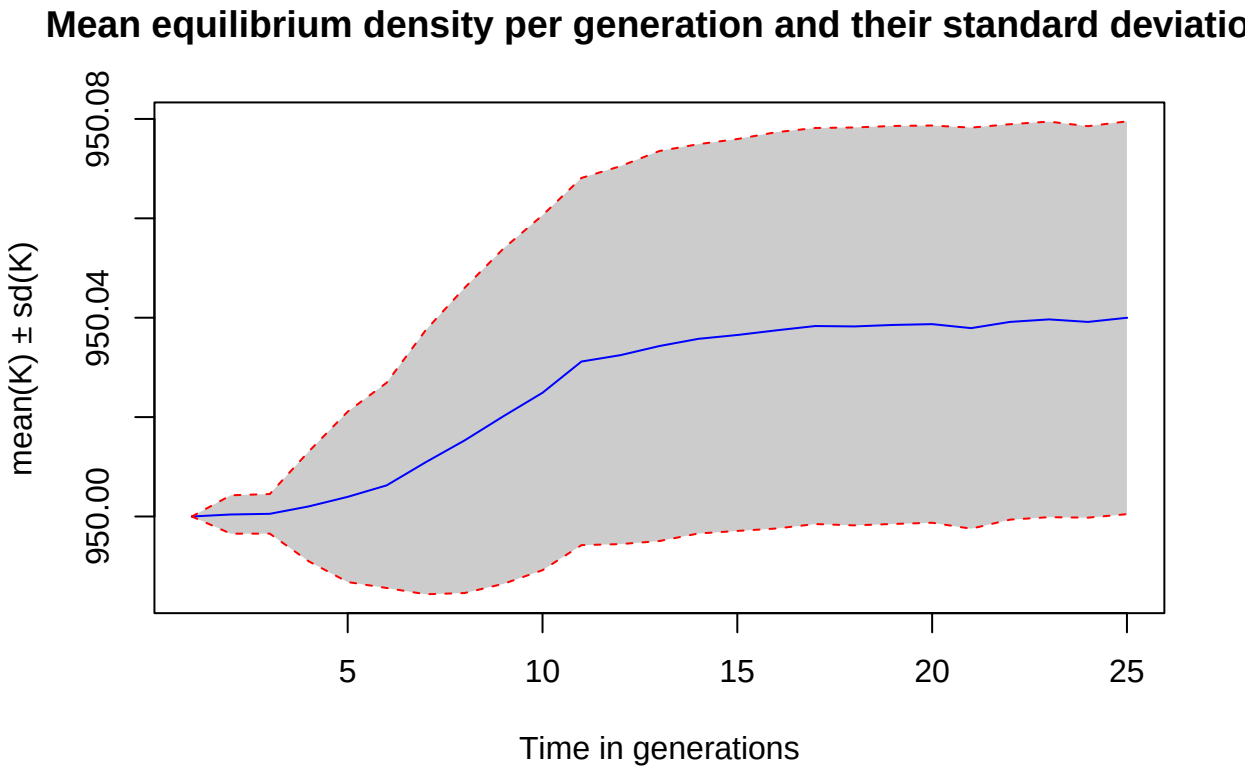


```
# plot logarithm of the relative density of population with mutation
plot(1:x, log10(gen$r), type="b", pch=20, col="blue", xlab="Time in generations", ylab="log10 of the rel
```

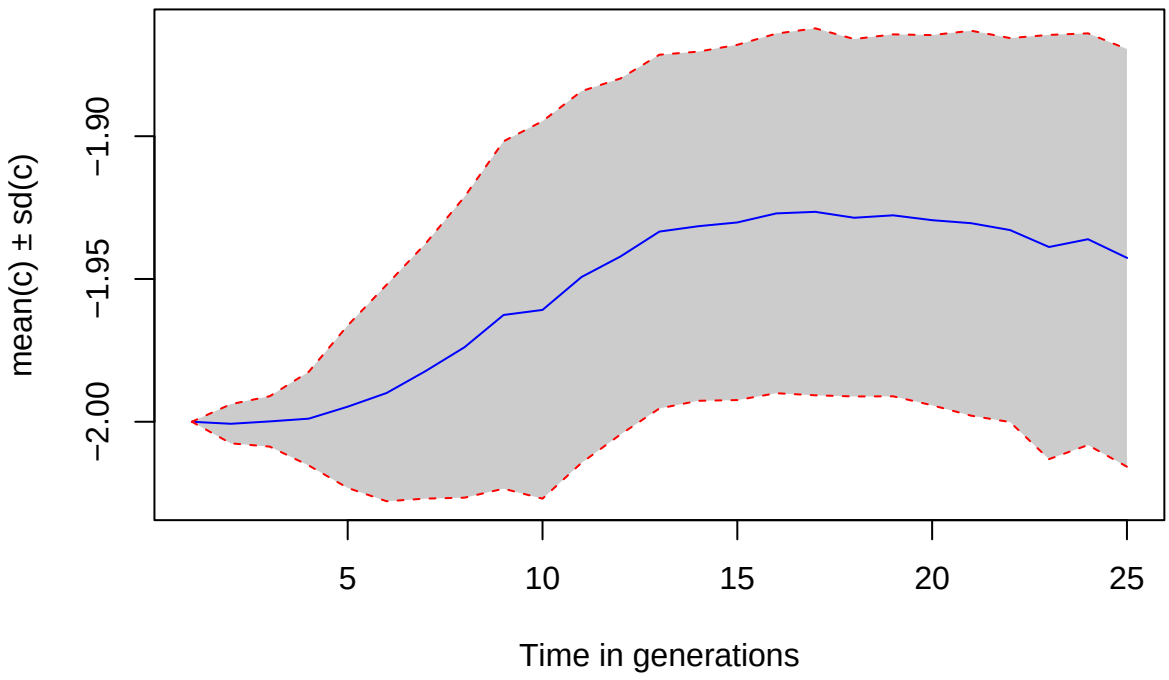


Comparing with the paper, we noticed that the relative density should eventually become equal to 1 for large times. Unfortunately, over the small time span we covered, this is not the case for us.

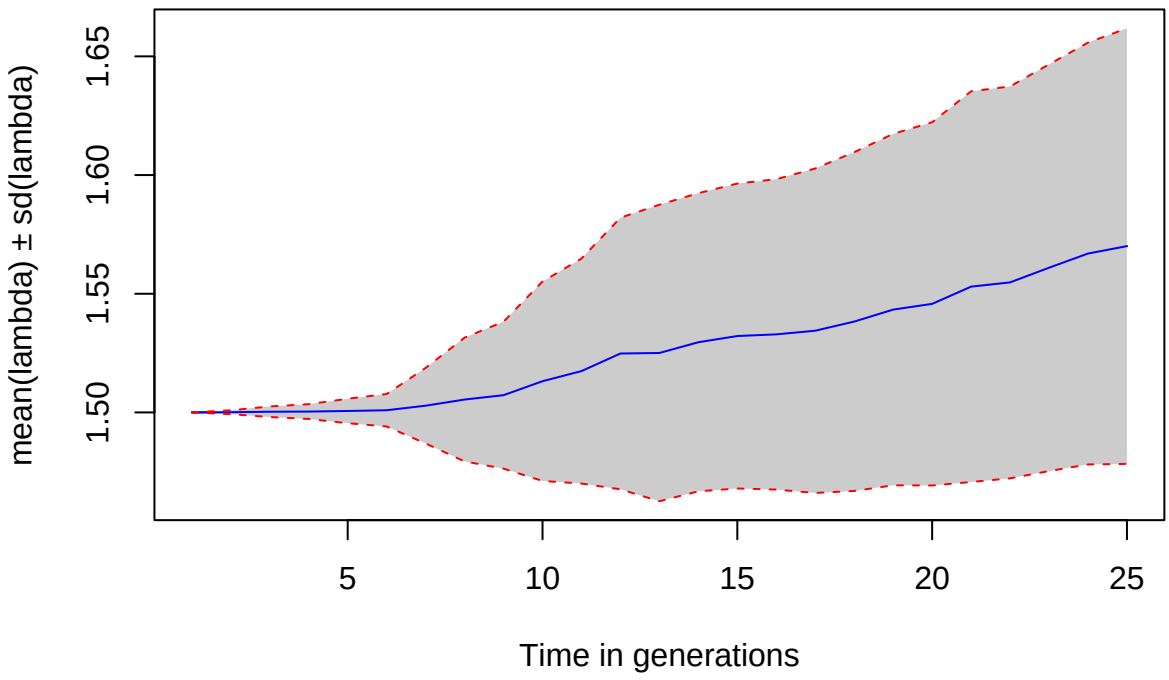
```
plot(1:x, gen$K_means, type = 'l', col="blue", ylim=c(min(gen$K_means-gen$K_sd), max(gen$K_means+gen$K_s
# add fill
polygon(c(x:1, 1:x), c(rev(gen$K_means-gen$K_sd), gen$K_means+gen$K_sd), col = 'grey80', border = NA)
lines(1:x, gen$K_means, col="blue")
# intervals
lines(1:x, gen$K_means-gen$K_sd, lty = 'dashed', col = 'red')
lines(1:x, gen$K_means+gen$K_sd, lty = 'dashed', col = 'red')
```



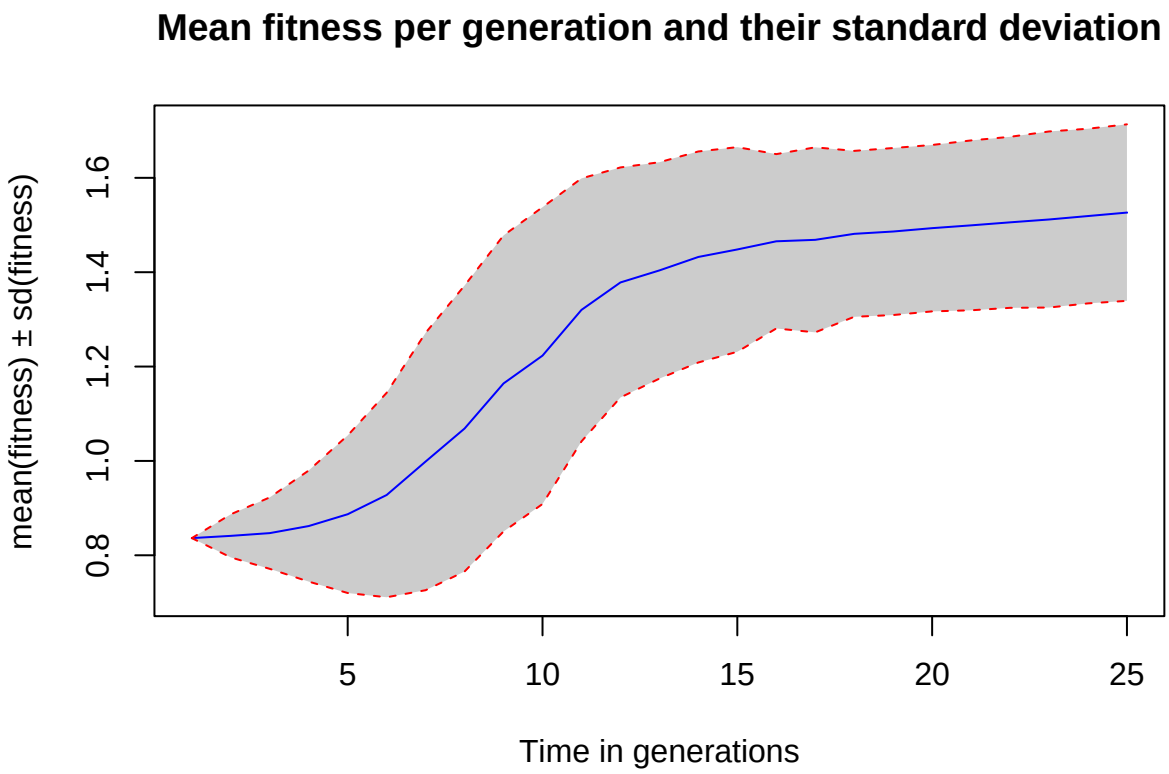

```
plot(1:x, gen$c_means, type = 'l', col="blue", ylim=c(min(gen$c_means-gen$c_sd), max(gen$c_means+gen$c_s
# add fill
polygon(c(x:1, 1:x), c(rev(gen$c_means-gen$c_sd), gen$c_means+gen$c_sd), col = 'grey80', border = NA)
lines(1:x, gen$c_means, col="blue")
# intervals
lines(1:x, gen$c_means-gen$c_sd, lty = 'dashed', col = 'red')
lines(1:x, gen$c_means+gen$c_sd, lty = 'dashed', col = 'red')
```




```
plot(1:x, gen$lambda_means, type = 'l', col="blue", ylim=c(min(gen$lambda_means-gen$lambda_sd), max(gen$
# add fill
polygon(c(x:1, 1:x), c(rev(gen$lambda_means-gen$lambda_sd), gen$lambda_means+gen$lambda_sd), col = 'grey
lines(1:x, gen$lambda_means, col="blue")
# intervals
lines(1:x, gen$lambda_means-gen$lambda_sd, lty = 'dashed', col = 'red')
lines(1:x, gen$lambda_means+gen$lambda_sd, lty = 'dashed', col = 'red')
```




```
plot(1:x, gen$fitness_means, type = 'l', col="blue", ylim=c(min(gen$fitness_means-gen$fitness_sd), max(g
# add fill
polygon(c(x:1, 1:x), c(rev(gen$fitness_means-gen$fitness_sd), gen$fitness_means+gen$fitness_sd), col =
lines(1:x, gen$fitness_means, col="blue")
# intervals
lines(1:x, gen$fitness_means-gen$fitness_sd, lty = 'dashed', col = 'red')
lines(1:x, gen$fitness_means+gen$fitness_sd, lty = 'dashed', col = 'red')
```


Mean fitness per generation and their standard deviation
mean(fitness) ± sd(fitness)
0.8
1.0
1.2
1.4
1.6
5
10
15
20
25
Time in generations