```r
library(mosaic)
library(R.utils)
library(Hmisc)

f <- makeFun(-sum(x ^ 4 - 16 * x ^ 2 + 5 * x) / 2 ~ x)


geneticAlgorithm <- function(fun, mi, pm, pc, tmax, limit, range) {
  P0 <- createPopulation(mi, range)
  t <- 0
  evaluation <- evaluate(fun, P0, range, limit)
  meanEvaluation <- matrix()
  bestInIteration <- matrix()
  best <- findBest(P0, evaluation)
  Pt <- P0
  while (t < tmax){
    meanEvaluation[t+1] <- mean(unlist(evaluation))
    bestInIteration[t+1] <- computeFunVal(f, best, range)
    R <- tournament(Pt, evaluation, mi, 2)
    C <- crossingOver(R, pc)
    M <- mutation(C, pm)
    evaluation <- evaluate(fun, M, range, limit)
    new_best <- findBest(M, evaluation)
    if (computeFunVal(f, best, range) < computeFunVal(f,new_best, range)){
      best <- new_best
    }
    Pt <- M
    t <- t + 1
  }
  bestVector <- best*range
  return(list(bestVector, meanEvaluation, bestInIteration))
}


computeFunVal <- function(fun, binary, range){
  which_numbers <- binary * range
  funVal <- fun(which_numbers)
  return(funVal)
}


evaluate <- function(fun, population, range, limit) {
  which_numbers <- lapply(population, function(x)
    x * range)
  y <- lapply(which_numbers, fun)
  num_elements <- lapply(population, sum)
  y[num_elements > limit] = 0
  y[num_elements < limit] = 0
  return(y)
}


findBest <- function(population, evaluation) {
  best_idx <- match(max(as.numeric(evaluation)), evaluation)
  best <- population[[best_idx]]
}


createPopulation <- function(mi, range) {
  set.seed(10)
  emptyPopulation <- vector(mode = "list", length = mi)
  population <- lapply(emptyPopulation,
                       function(x)
                         x <- round(runif(n = length(range))))
  set.seed(Sys.time())
  return(population)
}


tournament <- function(population, evaluation, mi, tournament_size) {
  new_population <- list()
  for (i in 1:mi) {
    rivals_idxs <- match(sample(population, tournament_size,
```

```
                                    replace = TRUE), population)
    rivals_evaluations <-
      lapply(rivals_idxs, function(x)
        evaluation[[x]])
    winner <-  max(as.numeric(rivals_evaluations))
    winner_idx <- rivals_idxs[[match(winner, rivals_evaluations)]]
    new_population[[i]] <- population[[winner_idx]]
  }
  return(new_population)
}

crossingOver <- function(population, pc){
  new_population <- list()
  population <- sample(population, length(population), replace = FALSE)
  for (i in 1:(length(population)/2)){
    chromosome1 <- population[[i]]
    chromosome2 <- population[[i+length(population)/2]]
    if (runif(1) < pc){
      crossPlace <- sample(c(1:(length(chromosome1)-1)), 1)
      new_chromosome1 <- c(chromosome1[1:crossPlace],
                           chromosome2[(crossPlace+1):length(chromosome2)])
      new_chromosome2 <- c(chromosome2[1:crossPlace],
                           chromosome1[(crossPlace+1):length(chromosome1)])
      new_population[[i]] <- new_chromosome1
      new_population[[i+length(population)/2]] <- new_chromosome2
    }
    else{
      new_population[[i]] <- chromosome1
      new_population[[i+length(population)/2]] <- chromosome2
    }
  }
  return(new_population)
}

mutation <- function(population, pm){
  new_population <- list()
  population <- sample(population, length(population), replace = FALSE)
  isMutated <- runif(length(population))<pm
  toMutate <- population[isMutated]
  new_population <- population[!isMutated]
  geneIdxs <- sample(c(1:length(population[[1]])), length(toMutate), replace = TRUE)
    if (length(toMutate)>0){
    for (i in 1:length(toMutate)){
      toMutate[[i]][[geneIdxs[i]]] <- !toMutate[[i]][[geneIdxs[i]]]
    }
  }
  new_population <- c(new_population, toMutate)
  return(new_population)
}

# testy
range <- c(-4:3)
populationSizes <- c(5,10,15,20,30,40,50,70,100)
bestVectors3 <- list()
meanEvaluations3 <- list()
bestsInIteration3 <- list()
for (i in 1:length(populationSizes)){
  result <- geneticAlgorithm(f, populationSizes[i], 0.1, 0.7, 500, 6, range)
  bestVectors3[[i]] <- result[[1]]
  meanEvaluations3[[i]] <- result[[2]]
  bestsInIteration3[[i]] <- result[[3]]
}

# ploty
for (i in 1:length(populationSizes)){
  png(file=paste("pop_size_",populationSizes[i],".png", collapse = NULL),
      width = 650, height = 450)
  plot(1:500, meanEvaluations3[[i]],
       main = paste("Średnia ocena pokolenia w funkcji numeru pokolenia dla
       rozmiaru populacji = ",populationSizes[i]),
       xlab = "Numer pokolenia",
       ylab = "Średnia ocena pokolenia")
  minor.tick(ny=10)
```

```
147     dev.off()
148   }
149
```