

Computational Statistics - Lab 06

Maximilian Pfundstein

2019-01-22

Contents

1	Question 1: Genetic Algorithm	1
2	Question 2: EM Algorithm	5
3	Source Code	6

1 Question 1: Genetic Algorithm

In this assignment, you will try to perform one-dimensional maximization with the help of a genetic algorithm.

Task: Define the function

$$f(x) = \frac{x^2}{e^x} - 2e^{\frac{-9\sin(x)}{x^2+x+1}}$$

```
f = function(x) {  
  left = x^2/exp(x)  
  exponent = (-9*sin(x))/(x^2+x+1)  
  right = 2*exp(exponent)  
  return(left-right)  
}
```

Task: Define the function `crossover()`: for two scalars `x` and `y` it returns their “kid” as $(x+y)/2$

```
crossover = function(x, y) return((x+y)/2)
```

Task: Define the function `mutate()` that for a scalar `x` returns the result of the integer division $x^2 \bmod 30$. (Operation mod is denoted in R as `%%`)

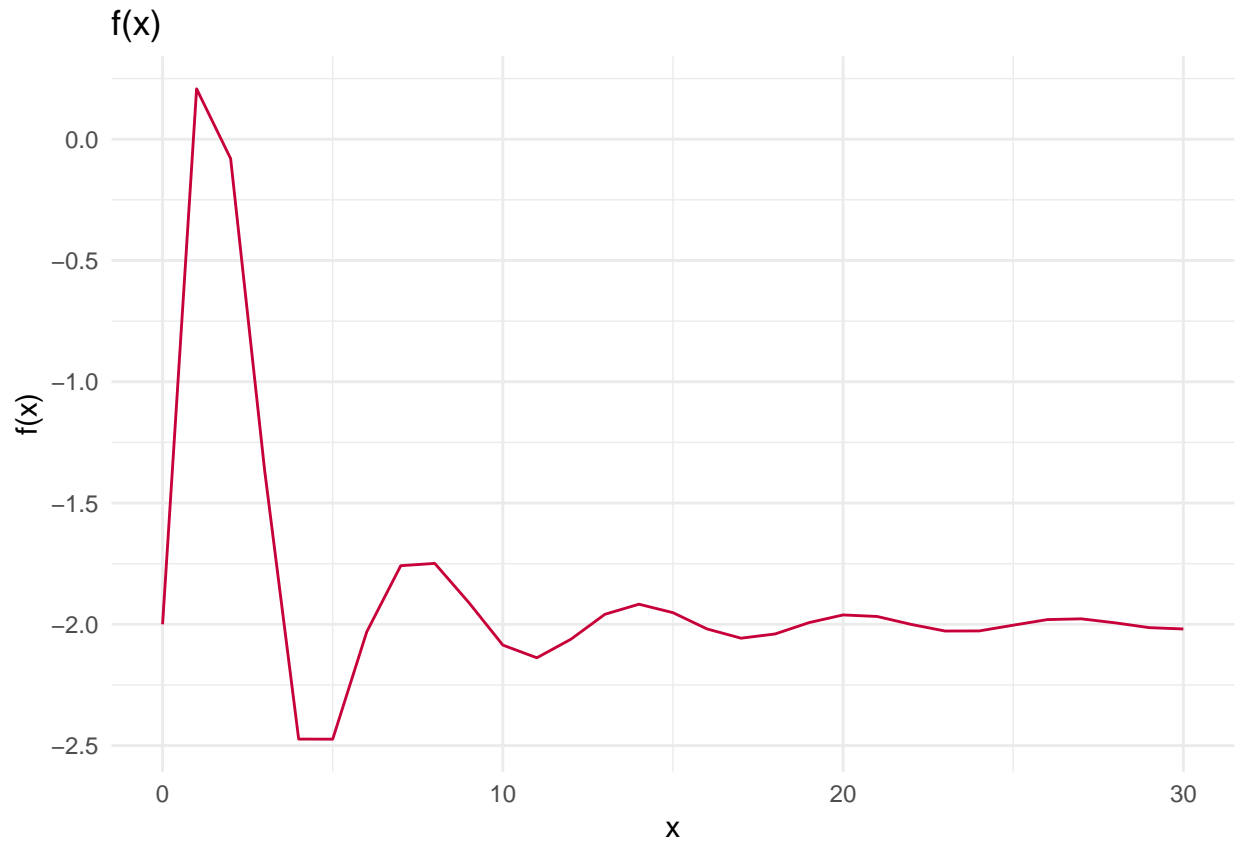
```
mutate = function(x) return((x^2)%%30)
```

Task: Write a function that depends on the parameters `maxiter` and `mutprob` and:

- Plots function `f` in the range from 0 to 30. Do you see any maximum value?
- Defines an initial population for the genetic algorithm as $X = (0, 5, 10, 15, \dots, 30)$
- Computes vector `Values` that contains the function values for each population point.
- Performs `maxiter` iterations where at each iteration
 - Two indexes are randomly sampled from the current population, they are further used as parents (use `sample()`).
 - One index with the smallest objective function is selected from the current population, the point is referred to as victim (use `order()`).
 - Parents are used to produce a new kid by crossover. Mutate this kid with probability `mutprob` (use `crossover()`, `mutate()`).
 - The victim is replaced by the kid in the population and the vector `Values` is updated.
 - The current maximal value of the objective function is saved.
- Add the final observations to the current plot in another colour.

Answer:

a: Let's have a look at the plot.



We see as maximum value at x:

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

With f(x):

```
print(max(f.sequence.))
```

```
## [1] 0.2076688
```

b: Let's define the initial population:

```
X = seq(from = 0, to = 30, by = 5)
```

c: Let us create the Values.

```
Values = f(X)
```

d: Let's create a for loop that is performing `maxiter` iterations as maximum.

```
genetic = function(X, Values, maxiter = 100, mutprob = 0.05) {  
  best_individual = NaN  
  for (i in 1:maxiter) {  
    # 1)
```

```

parents = sample(1:length(X), size = 2)

# 2) I don't know why we should order here!?
victim = which.min(Values)

# 3)
child = crossover(X[parents][1], X[parents][2])
if (mutprob > runif(n = 1, min = 0, max = 1)) {
  child = mutate(child)
}

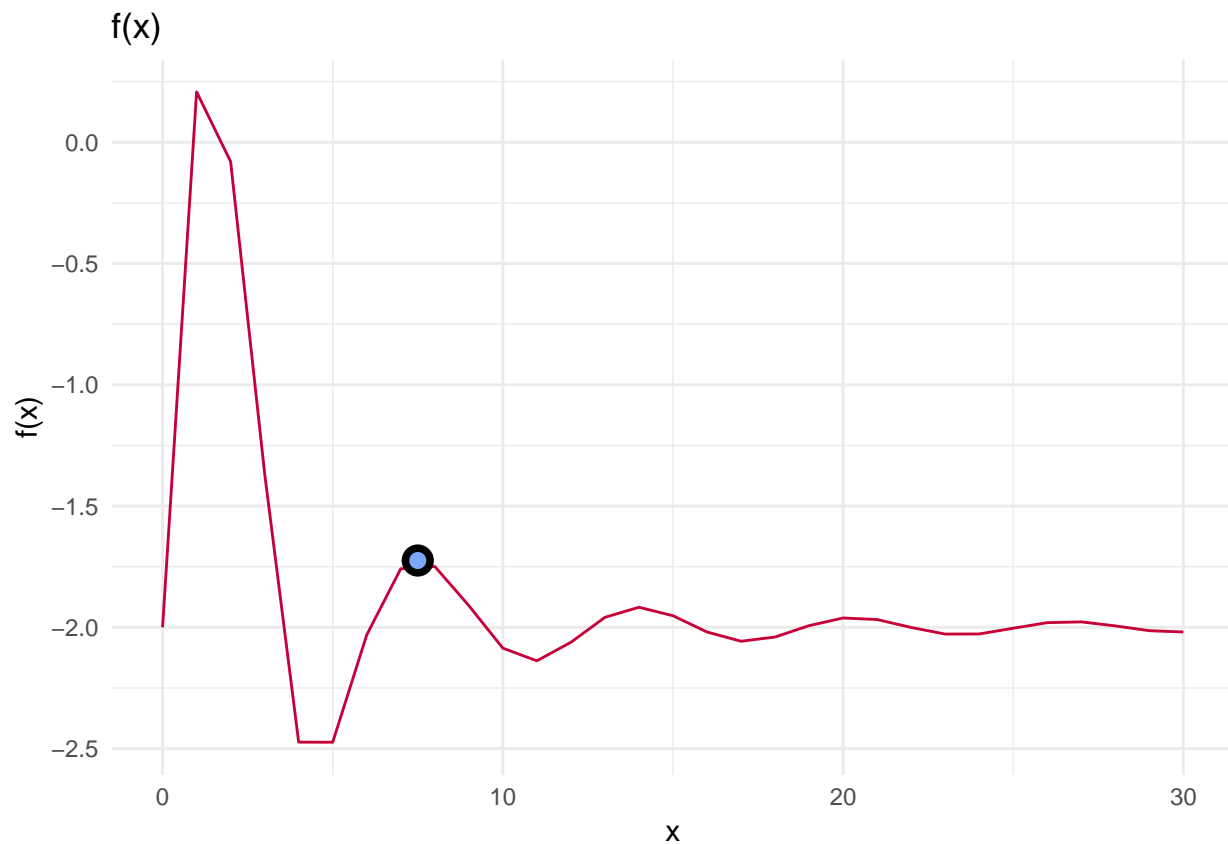
# 4)
X[victim] = child
Values[victim] = f(child)

# 5)
best_index = which.max(Values)
best_individual = list(cx = X[best_index], cy = Values[best_index])
}
return(list(best = best_individual, population = X))
}

best_individual = genetic(X, Values, 100, 0.05)$best

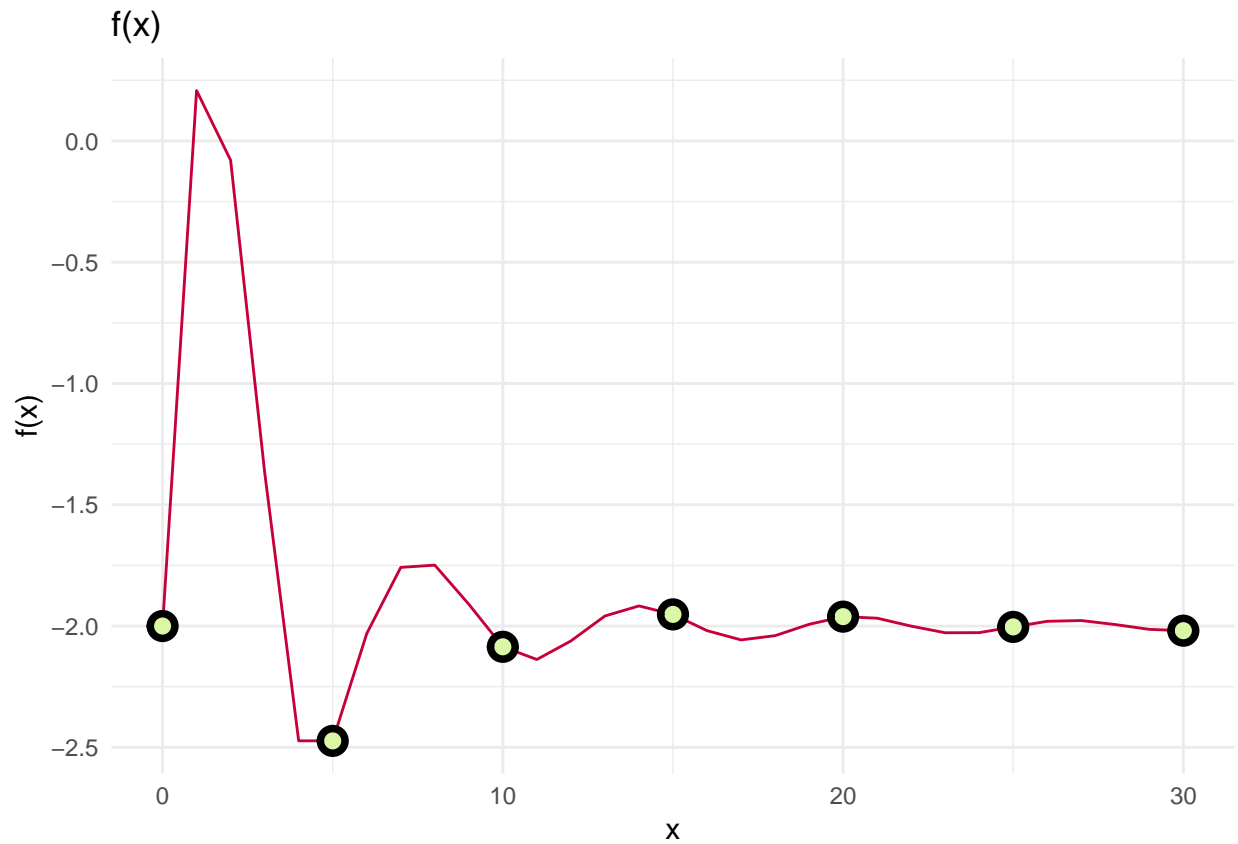
```

e: Let's add the final (best) value to the plot.

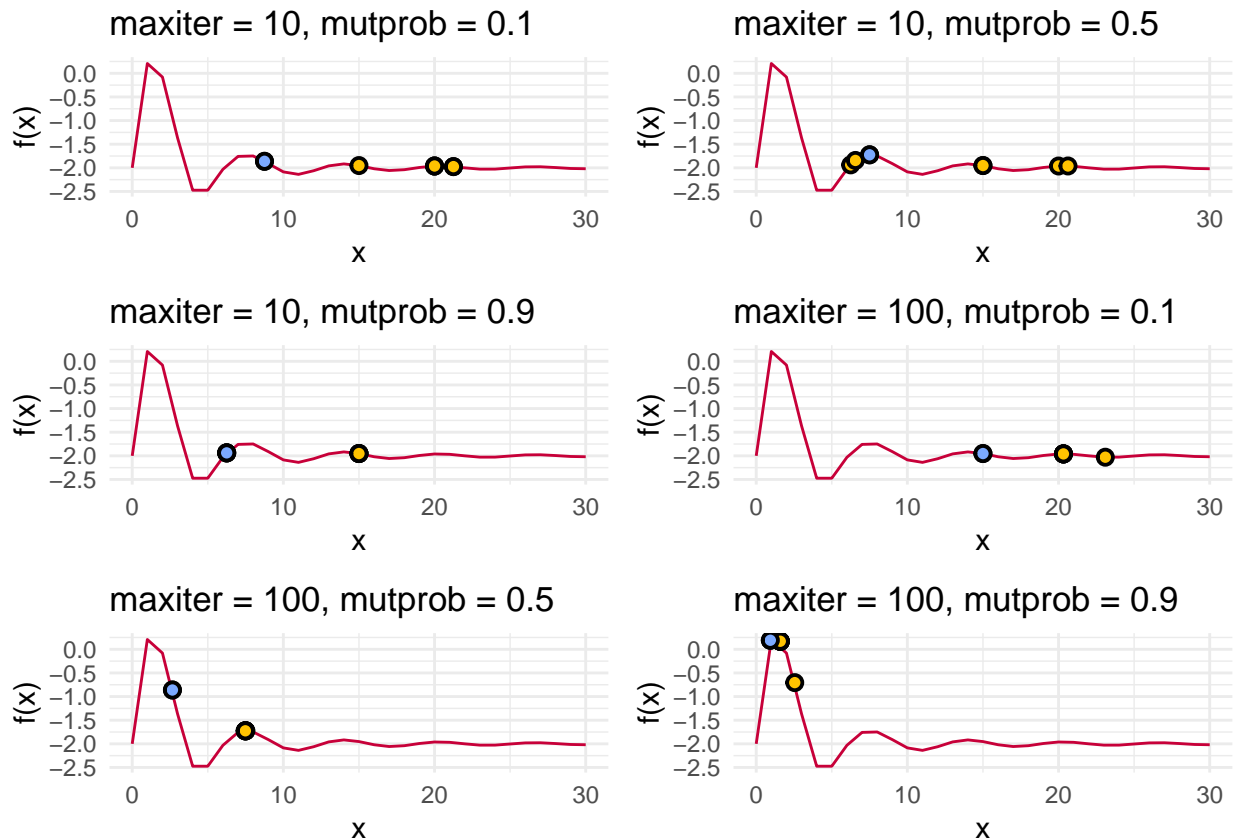


Task: Run your code with different combinations of `maxiter` = 10, 100 and `mutprob`= 0.1, 0.5, 0.9. Observe the initial population and final population. Conclusions?

Answer: The initial distribution looks like this:



In the following you see the plots for the different combinations of parameter for a seed of 12345. Keep in mind that these pictures and also the conclusions might change with a different seed or reruns.



As we can see we sometimes find the (nearly) best solution, but sometimes we also get stuck in a local maximum or run out of iterations to further climb the hill. We therefore conclude, that genetic (or evolutionary algorithms) are a nice way to scrape a large feature space, but that there are problems which have to be handled. There are different ways to do the mutation, crossover and the selection, also we can define a learning to prevent the algorithm getting stuck too early. Keep in mind that we here only take the best individual in the current iteration, not the overall best one, so we might “delete” some better solutions here (at least that’s how it is in my implementation at the moment). So in teh end, like most models, the hyperparameters have a large impact on the performance of our algorithm.

From previous studies with the topic we know that the way of crossover, mutation and selection have a **great** impact on the performance and can be optimized doing bitwise operations (where the significacne of bits is also important). Also the whole process can be made faster if a different language and multiprocessing (or even a GPU) are taken into consideration.

2 Question 2: EM Algorithm

The data file `physical.csv` describes a behavior of two related physical processes $Y = Y(X)$ and $Z = Z(X)$.

Task: Make a time series plot describing dependence of Z and Y versus X . Does it seem that two processes are related to each other? What can you say about the variation of the response values with respect to X ?

Task: Note that there are some missing values of Z in the data which implies problems in estimating models by maximum likelihood. Use the following model

$$Y_i \sim e^{x_i/\lambda}, \quad Z_i \sim e^{x_i/2\lambda}$$

where λ is some unknown parameter. **The goal is to derive an EM algorithm that estimates λ .**

Task: Implement this algorithm in R, use $\lambda_0 = 100$ and convergence criterion “stop if the change in λ is less than 0.001”. What is the optimal λ and how many iterations were required to compute it?

Task: Plot $E[Y]$ and $E[Z]$ versus X in the same plot as Y and Z versus X . Comment whether the computed λ seems to be reasonable.

3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = TRUE, include = TRUE, eval = TRUE)
library(ggplot2)
library(knitr)
library(gridExtra)

f = function(x) {
  left = x^2/exp(x)
  exponent = (-9*sin(x))/(x^2+x+1)
  right = 2*exp(exponent)
  return(left-right)
}

crossover = function(x, y) return((x+y)/2)

mutate = function(x) return((x^2)%30)

sequence = seq(from = 0, to = 30, by = 1)
f.sequence. = f(sequence)
df = data.frame(sequence, f.sequence.)

ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  labs(title = "f(x)", y = "f(x)", x = "x") +
  theme_minimal()

print(sequence[which.max(f.sequence.)])

print(max(f.sequence.))

X = seq(from = 0, to = 30, by = 5)

Values = f(X)

genetic = function(X, Values, maxiter = 100, mutprob = 0.05) {
```

```

best_individual = NaN

for (i in 1:maxiter) {

  # 1)
  parents = sample(1:length(X), size = 2)

  # 2) I don't know why we should order here!?
  victim = which.min(Values)

  # 3)
  child = crossover(X[parents][1], X[parents][2])
  if (mutprob > runif(n = 1, min = 0, max = 1)) {
    child = mutate(child)
  }

  # 4)
  X[victim] = child
  Values[victim] = f(child)

  # 5)
  best_index = which.max(Values)
  best_individual = list(cx = X[best_index], cy = Values[best_index])
}
return(list(best = best_individual, population = X))
}

best_individual = genetic(X, Values, 100, 0.05)$best

df = data.frame(sequence, f.sequence.)
best_individual = as.data.frame(best_individual)

ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  geom_point(aes(x = cx, y = cy), data = best_individual, color = "black",
    fill = "#7BA9FF", shape = 21, size = 3, stroke = 2) +
  labs(title = "f(x)", y = "f(x)", x = "x") +
  theme_minimal()

df = data.frame(sequence, f.sequence.)
initial.population = data.frame(X, Values)

ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  geom_point(aes(x = X, y = Values), data = initial.population, color = "black",
    fill = "#DAF7A6", shape = 21, size = 3, stroke = 2) +
  labs(title = "f(x)", y = "f(x)", x = "x") +
  theme_minimal()

set.seed(12345)

```

```

genetic1 = genetic(X, Values, 10, 0.1)
genetic2 = genetic(X, Values, 10, 0.5)
genetic3 = genetic(X, Values, 10, 0.9)
genetic4 = genetic(X, Values, 100, 0.1)
genetic5 = genetic(X, Values, 100, 0.5)
genetic6 = genetic(X, Values, 100, 0.9)

population1 = data.frame(genetic1$population, f(genetic1$population))
best1 = as.data.frame(genetic1$best)

population2 = data.frame(genetic2$population, f(genetic2$population))
best2 = as.data.frame(genetic2$best)

population3 = data.frame(genetic3$population, f(genetic3$population))
best3 = as.data.frame(genetic3$best)

population4 = data.frame(genetic4$population, f(genetic4$population))
best4 = as.data.frame(genetic4$best)

population5 = data.frame(genetic5$population, f(genetic5$population))
best5 = as.data.frame(genetic5$best)

population6 = data.frame(genetic6$population, f(genetic6$population))
best6 = as.data.frame(genetic6$best)

p1 = ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  geom_point(aes(x = genetic1.population, y = f.genetic1.population.), data = population1, color = "black",
    fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
  geom_point(aes(x = cx, y = cy), data = best1, color = "black",
    fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
  labs(title = "maxiter = 10, mutprob = 0.1", y = "f(x)", x = "x") +
  theme_minimal()

p2 = ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  geom_point(aes(x = genetic2.population, y = f.genetic2.population.), data = population2, color = "black",
    fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
  geom_point(aes(x = cx, y = cy), data = best2, color = "black",
    fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
  labs(title = "maxiter = 10, mutprob = 0.5", y = "f(x)", x = "x") +
  theme_minimal()

p3 = ggplot(df) +
  geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
  geom_point(aes(x = genetic3.population, y = f.genetic3.population.), data = population3, color = "black",
    fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
  geom_point(aes(x = cx, y = cy), data = best3, color = "black",
    fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
  labs(title = "maxiter = 10, mutprob = 0.9", y = "f(x)", x = "x") +
  theme_minimal()

p4 = ggplot(df) +

```



```

geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
geom_point(aes(x = genetic4.population, y = f.genetic4.population.), data = population4, color = "black",
            fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
geom_point(aes(x = cx, y = cy), data = best4, color = "black",
            fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
labs(title = "maxiter = 100, mutprob = 0.1", y = "f(x)", x = "x") +
theme_minimal()

p5 = ggplot(df) +
geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
geom_point(aes(x = genetic5.population, y = f.genetic5.population.), data = population5, color = "black",
            fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
geom_point(aes(x = cx, y = cy), data = best5, color = "black",
            fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
labs(title = "maxiter = 100, mutprob = 0.5", y = "f(x)", x = "x") +
theme_minimal()

p6 = ggplot(df) +
geom_line(aes(x = sequence, y = f.sequence.), color = "#C70039") +
geom_point(aes(x = genetic6.population, y = f.genetic6.population.), data = population6, color = "black",
            fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
geom_point(aes(x = cx, y = cy), data = best6, color = "black",
            fill = "#7BA9FF", shape = 21, size = 2, stroke = 1) +
labs(title = "maxiter = 100, mutprob = 0.9", y = "f(x)", x = "x") +
theme_minimal()

grid.arrange(p1, p2, p3, p4, p5, p6, nrow = 3)

```