Computational Statistics - Lab 06

Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)

2019-02-02

Contents

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

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

```
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 $x^2 mod 30$. (Operation mod is denoted in R as `%%`)

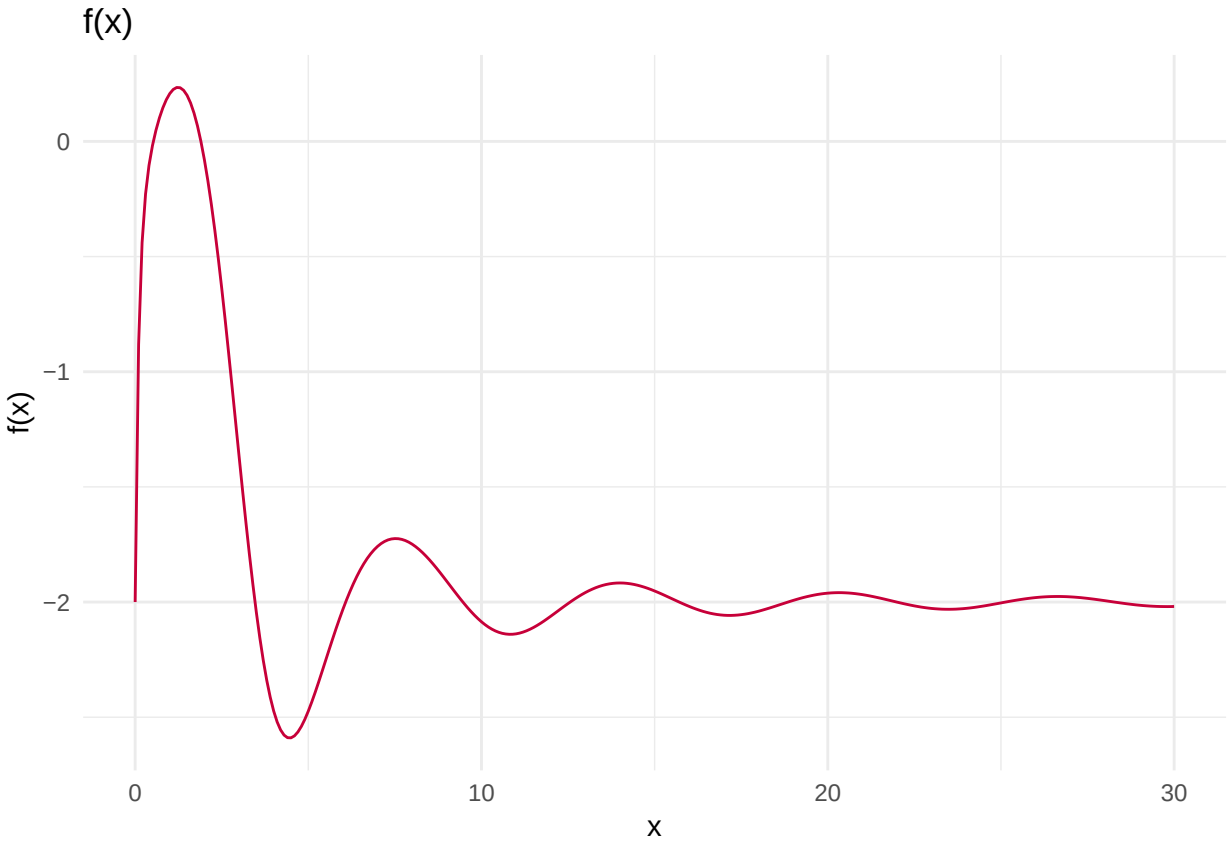
```
mutate = function(x) return((x^2)%%30)
```

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

a. Plots function `f` in the range from 0 to 30. Do you see any maximum value?
b. Defines an initial population for the genetic algorithm as $X = (0, 5, 10, 15, ..., 30)$
c. Computes vector `Values` that contains the function values for each population point.
d. Performs `maxiter` iterations where at each iteration
    1. Two indexes are randomly sampled from the current population, they are further used as parents (use `sample()`).
    2. One index with the smallest objective function is selected from the current population, the point is referred to as victim (use `order()`).
    3. Parents are used to produce a new kid by crossover. Mutate this kid with probability `mutprob` (use `crossover()`, `mutate()`).
    4. The victim is replaced by the kid in the population and the vector `Values` is updated.
    5. The current maximal value of the objective function is saved.
e. 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] 1.2
```

With `f(x)`:

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

```
## [1] 0.2341007
```

`b:` Let's define the initial population:

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

`c:` Let us create the `Values`.

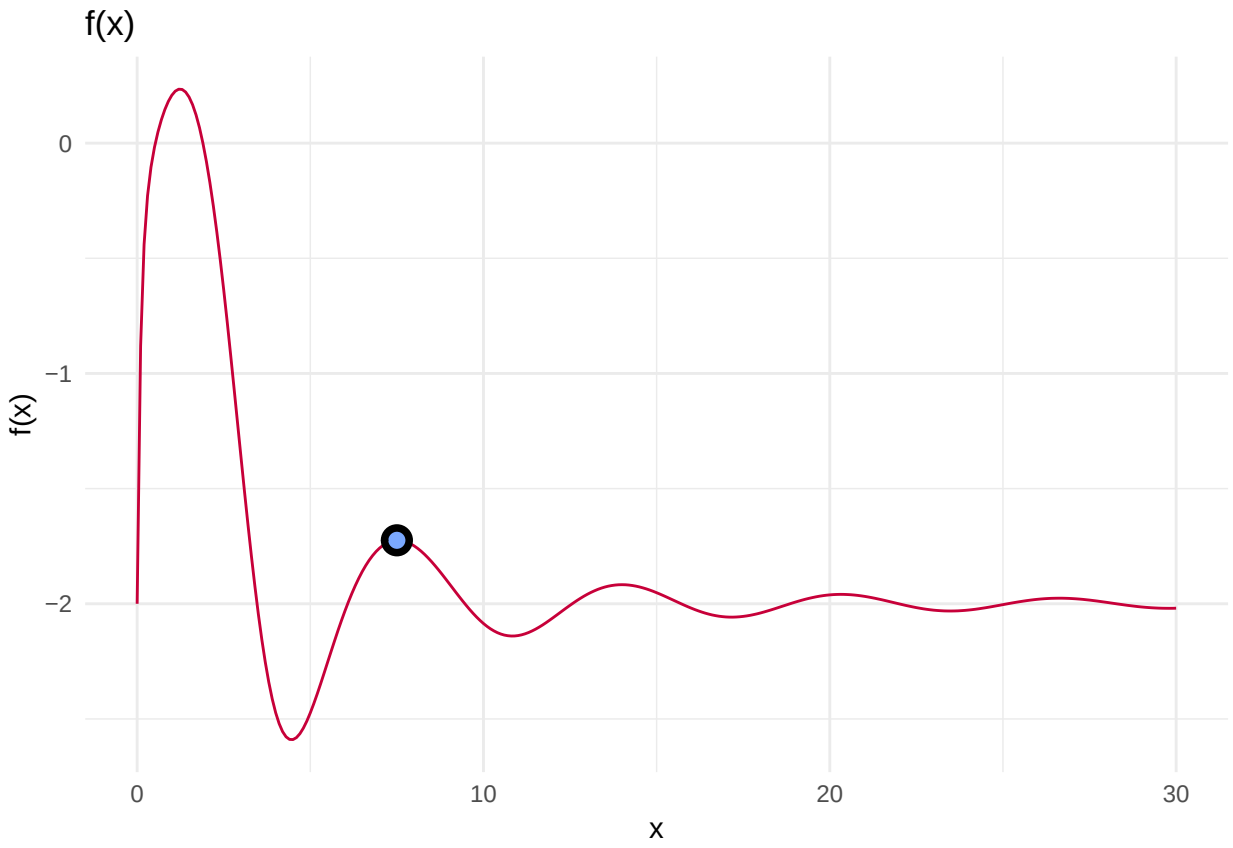
```
Values = f(X)
```

`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)
```

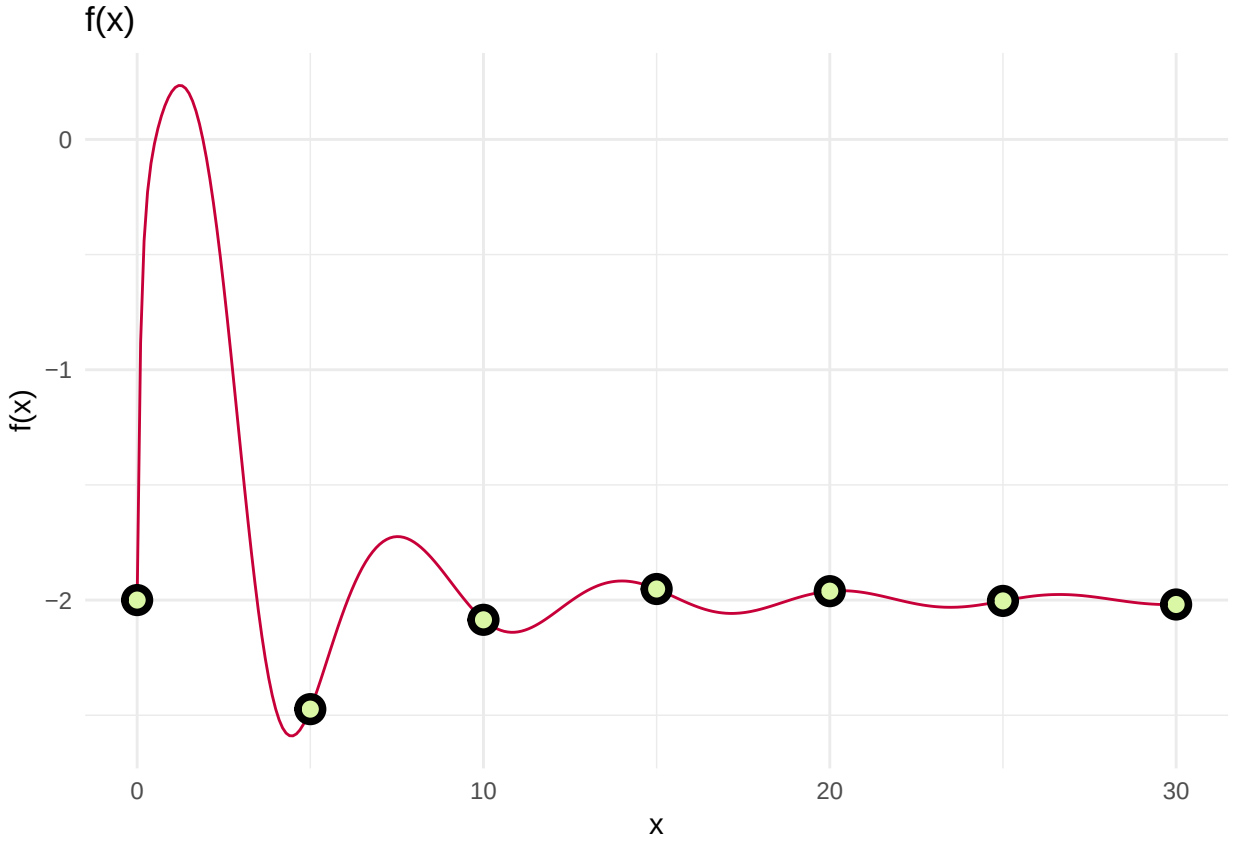
```
    # 2) I don't know why we should order here!?
    victim = which.min(Values)

    # 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.

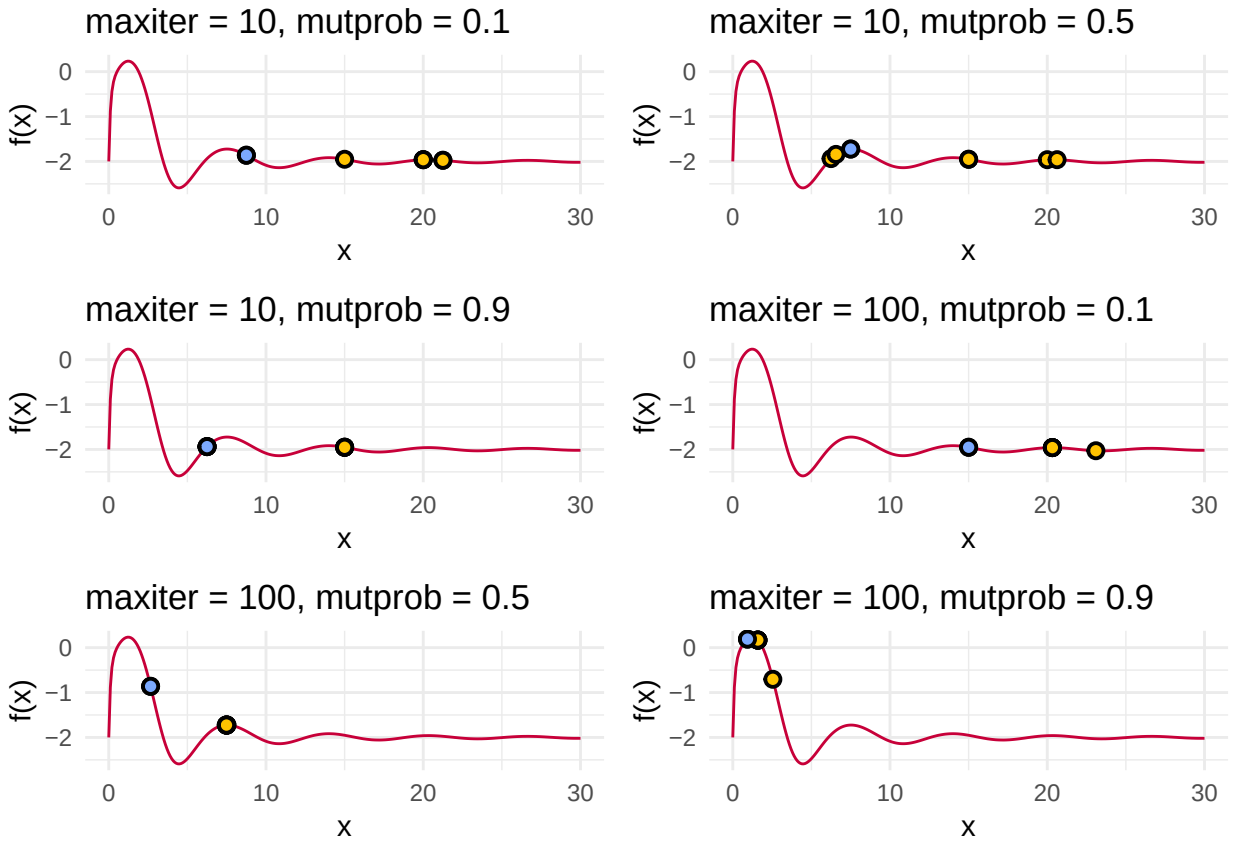


Task: Run your code with different combinations of `maxiter = 10, 100` and `mutprob= 0.1, 0.5, 0.9`. Observe the initial population and final population. Conclusions?

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 trailing learning rate to prevent the algorithm getting stuck too early. Keep in mind that here we 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 our implementation at the moment). So in the end, like most models, the hyperparameters have a large impact on the performance of our algorithm.

From previous studies within the topic we know that the way of crossover, mutation and selection have a **great** impact on the performance and can be optimized doing bitvise operations (where the significance 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 $Z = Z(X)$.

```
##     X         Y         Z
## 1 1.1  5.414260 11.987697
## 2 1.2  4.226085  4.556510
## 3 1.3  9.678230 26.866863
## 4 1.4  7.566957 19.909734
## 5 1.5  7.588824 27.454942
## 6 1.6 12.331576  2.083665
```

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?

Answer: Yes it seems like that those two variables are related to each other as they have spikes in the same areas. It ssems like that we have some kind of cyclical behavior. The amplitude of our spikes decreases with increasing X. It looks like a damped oscillation.

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 unkown parameter. **The goal is to derive an EM algorithm that estimates λ.**

Answer: For deriving the EM algorithm we have to iteratively perform the expectation (E) and maximization (M) step. The E-step consists of calculating:

E-Step Explanation:

$$Q(\theta, \theta^k) = E[\mathcal{L}(\theta|Y, Z)|\theta^k, Y]$$

In this example the unkown parameter is $\theta = \lambda$. The distribution Z_i should now be confused with λ as this, as the unkown parameter, is usually referenced as Z as well in the literature. After that, we move on to the M-Step.

M-Step Explanation:

Here we maximize $Q(\lambda, \lambda^k)$ by taking the derivative and setting it so 0.

It's time to calculate those values.

E-Step Calculation:

Both Y_i and Z_i behave like a exponential distribution with the following λs.

$$\lambda_Y = \frac{X_i}{\lambda}, \quad \lambda_Z = \frac{X_i}{2\lambda}$$

So we have the following distributions:

$$Y_i \sim \frac{X_i}{\lambda} e^{-X_i/\lambda}, \quad Z_i \sim \frac{X_i}{2\lambda} e^{-X_i/2\lambda}$$

The log likelihood is the log of the product of the two distributions and thus given by:

$$\mathcal{L}(\lambda|Y,Z) = ln\left(\prod_{i=1}^{n} \frac{X_i}{\lambda} e^{-X_i/\lambda} * \frac{X_i}{2\lambda} e^{-X_i/2\lambda}\right)$$

We take out the variables that do not depend on the product:

$$\mathcal{L}(\lambda|Y,Z) = ln\left(\left(\frac{1}{2\lambda^2}\right)^n \prod_{i=1}^{n} X_i^2 e^{\frac{1}{\lambda}(X_i Y_i + \frac{X_i}{2} Z_i)}\right)$$

Then we resolve the log:

$$\mathcal{L}(\lambda|Y,Z) = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - \sum_{i=1}^{n} \frac{1}{\lambda}\left(X_i Y_i + \frac{X_i}{2} Z_i\right)$$

Next we have to calculate the expected value of or $\mathcal{L}(\lambda|Y,Z)$ as we have latent variables which we don't know even after the event. As only those are missing, we only need to take the expected value of the latent varibles, which are in Z_i:

$$E[\mathcal{L}(\lambda|Y,Z)] = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - E\left[\sum_{i=1}^{n} \frac{1}{\lambda}\left(X_i Y_i + \frac{X_i}{2} Z_i\right)\right]$$

$$E[\mathcal{L}(\lambda|Y,Z)] = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - \sum_{i=1}^{n} \frac{X_i Y_i}{\lambda} - E\left[\sum_{i=1}^{n} \frac{X_i Z_i}{2\lambda}\right]$$

Now we need to further separate our Z_i's as they contain known and unkown values of which we have to calculate the expected value. We assume the set Z_i to be ordered, where the known variables come first and the latent variables appear after that. We will use the index β to denote the last known variables. Therefore the first latent variable is denoted by $\beta + 1$. To make it clearer which variables are meant, in addition to the index, we introduce Λ as the subset of z that only contains teh latent variables and is indexed by $\beta + 1$ to n. Taking this into account, we continue.

$$E[\mathcal{L}(\lambda|Y,Z)] = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - \sum_{i=1}^{n} \frac{X_i Y_i}{\lambda} - \sum_{i=1}^{\beta} \frac{X_i Z_i}{2\lambda} - \sum_{i=\beta+1}^{n} \frac{X_i E[\Lambda_i]}{2\lambda}$$

As $E[\Lambda_i]$ follows the exponential distribution the expected value is given by $\frac{1}{\lambda_k}$.

$$E[\Lambda_i] = \frac{1}{\lambda_k} = \frac{1}{\frac{X_i}{2\lambda_k}} = \frac{2\lambda_k}{X_i}$$

Inserting this into the above formula we get:

$$E[\mathcal{L}(\lambda|Y,Z)] = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - \sum_{i=1}^{n} \frac{X_i Y_i}{\lambda} - \sum_{i=1}^{\beta} \frac{X_i Z_i}{2\lambda} - \sum_{i=\beta+1}^{n} \frac{\lambda_k}{\lambda}$$

And finally we get:

$$Q(\lambda, \lambda^k) = E[\mathcal{L}(\lambda|Y,Z)] = -n * ln(2\lambda^2) + 2\sum_{i=1}^{n} ln(X_i) - \sum_{i=1}^{n} \frac{X_i Y_i}{\lambda} - \sum_{i=1}^{\beta} \frac{X_i Z_i}{2\lambda} - (n - \beta\frac{\lambda_k}{\lambda})$$

M-Step Calculation:

To find the new λ which maximizes the likelihood given the expected variables we need to take the derivative in respect to λ, set the equation equal to 0 and solve for λ.

$$\frac{\partial Q(\lambda, \lambda^k)}{\partial \lambda} = -\frac{2n}{\lambda} + \sum_{i=1}^{n} \frac{X_i Y_i}{\lambda^2} + \frac{1}{2}\sum_{i=1}^{\beta} \frac{X_i Z_i}{\lambda^2} + (n - \beta)\frac{\lambda_k}{\lambda^2}$$

Setting this to 0 and solving for λ we receive:

$$\lambda = \frac{\sum_{i=1}^{n} X_i Y_i + \frac{1}{2}\sum_{i=1}^{\beta} X_i Z_i + (n - \beta)\lambda_k}{2n}$$

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?

```r
# To keep it simple we will not pass the data to the function.
lambda_estimate_em = function (input_iterations = 100, input_treshold = 0.0001) {

  iterations_max = input_iterations
  iterations = 0
  threshold = input_treshold

  n = nrow(data)
  lambdas = c()
  lambda = NaN
  lambda_k = 100

  Z = data$Z[!is.na(data$Z)]
  A = data$Z[is.na(data$Z)]

  Z_index =  which(!is.na(data$Z))
  A_index =  which(is.na(data$Z))

  X = data$X[Z_index]
  X_A = data$X[A_index]

  beta = length(Z)

  for (i in 1:iterations_max) {
    iterations = iterations + 1
```

```
    # E/M-Step
    lambda = lambda_k
    lambda_k = (sum(data$X * data$Y) + 0.5 * sum(X * Z) +
                  ((n - beta) * lambda_k)) / (2 * n)
    lambdas = c(lambdas, lambda_k)

    if (abs(lambda_k - lambda) < threshold ) break
  }

  return(list(lambdas, iterations))
}

em_result = lambda_estimate_em()
lambda_result = em_result[[1]][em_result[[2]]]

print(em_result)
```

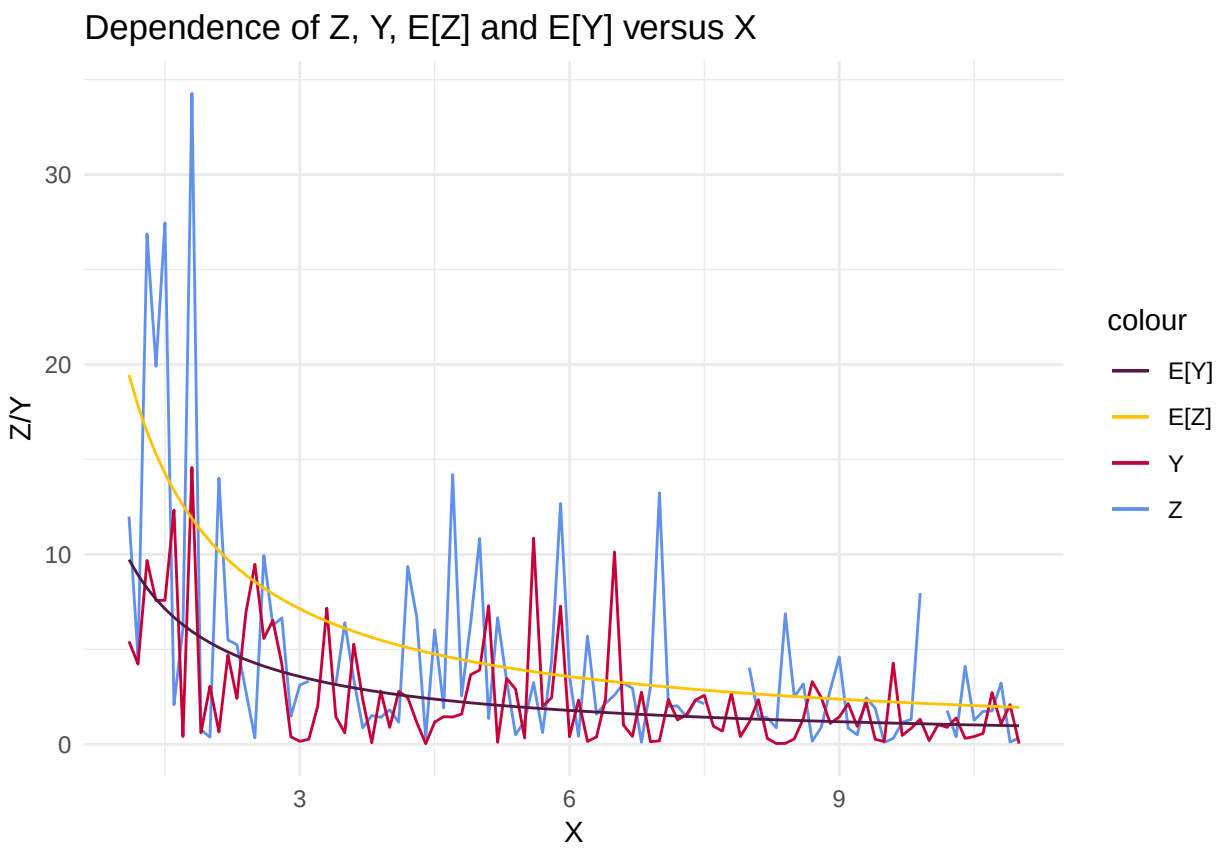
```
## [[1]]
## [1] 14.26782 10.83853 10.70136 10.69587 10.69566 10.69565
##
## [[2]]
## [1] 6
```

So we see that it took 6 iterations to find the optimal $\lambda = 10.69565$.

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.

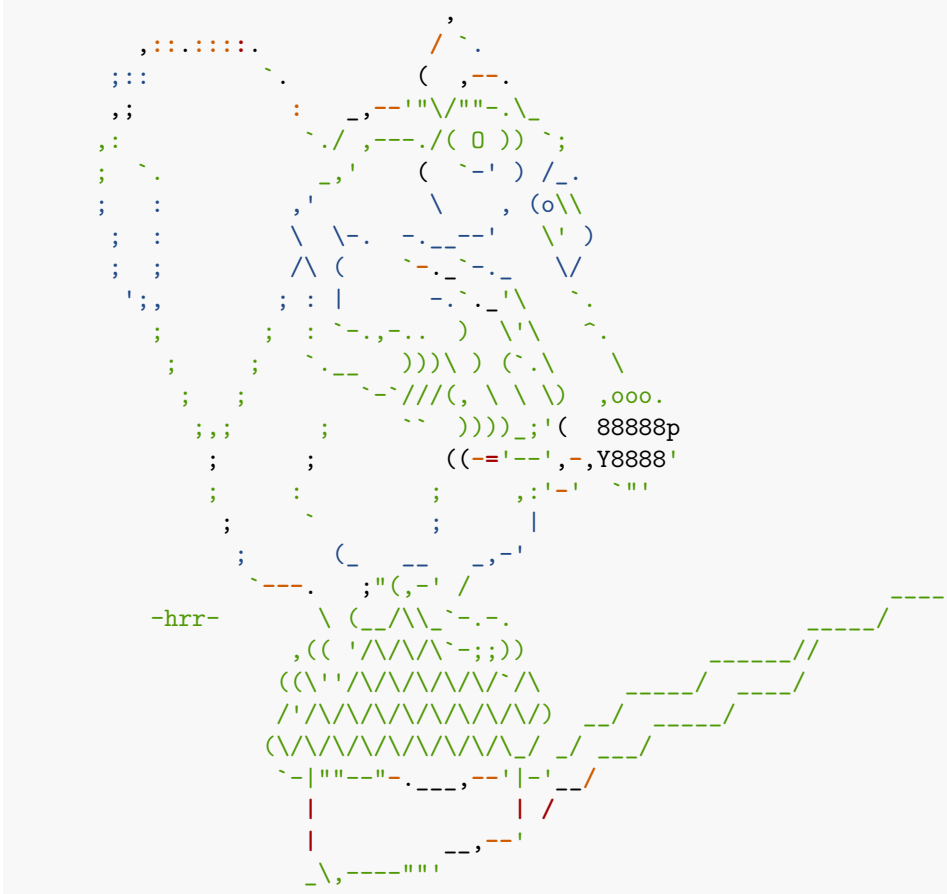
Answer: Given the exponential distribution the expected values are caluclated by the following:

$$E[Y] = \frac{\lambda}{X_i}, \quad E[Y] = \frac{2\lambda}{X_i}$$



The value for the computed λ seems reasonable as the expected values follow the general trend of the data.

```
Nüffe, wo sind meine Nüffe?
```



3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, 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)
```

```r
sequence = seq(from = 0, to = 30, by = 0.1)
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 = "bla
             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 = "bla
             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 = "bla
             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 = "bla
             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 = "bla
             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 = "bla
             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)
```

```r
data = read.csv("physical1.csv", sep = ",", dec = ".")
head(data)


ggplot(data) +
  geom_line(aes(x = X, y = Z, color = "Z")) +
  geom_line(aes(x = X, y = Y, color = "Y")) +
  labs(title = "Dependence of Z and Y versus X", y = "Z/Y", x = "X") +
  scale_color_manual(values = c("#C70039", "#581845")) +
  theme_minimal()


# To keep it simple we will not pass the data to the function.
lambda_estimate_em = function (input_iterations = 100, input_treshold = 0.0001) {

  iterations_max = input_iterations
  iterations = 0
  threshold = input_treshold

  n = nrow(data)
  lambdas = c()
  lambda = NaN
  lambda_k = 100

  Z = data$Z[!is.na(data$Z)]
  A = data$Z[is.na(data$Z)]

  Z_index =  which(!is.na(data$Z))
  A_index =  which(is.na(data$Z))

  X = data$X[Z_index]
  X_A = data$X[A_index]

  beta = length(Z)

  for (i in 1:iterations_max) {
    iterations = iterations + 1

    # E/M-Step
    lambda = lambda_k
    lambda_k = (sum(data$X * data$Y) + 0.5 * sum(X * Z) +
                  ((n - beta) * lambda_k)) / (2 * n)
    lambdas = c(lambdas, lambda_k)

    if (abs(lambda_k - lambda) < threshold ) break
  }

  return(list(lambdas, iterations))
}

em_result = lambda_estimate_em()
lambda_result = em_result[[1]][em_result[[2]]]
```

```r
print(em_result)


data$Y_E = (lambda_result)/data$X
data$Z_E = (2*lambda_result)/data$X

ggplot(data) +
  geom_line(aes(x = X, y = Z, color = "Z")) +
  geom_line(aes(x = X, y = Y, color = "Y")) +
    geom_line(aes(x = X, y = Z_E, color = "E[Z]")) +
  geom_line(aes(x = X, y = Y_E, color = "E[Y]")) +
  labs(title = "Dependence of Z, Y, E[Z] and E[Y] versus X", y = "Z/Y", x = "X") +
  scale_color_manual(values = c("#581845", "#FFC300", "#C70039", "#6091ec")) +
  theme_minimal()
```

Nüffe, wo sind meine Nüffe?