Lab06

Shipeng Liu (shili506) Dongwei Ni (donni508)

2022-12-10

Question 1: Genetic algorithm

##1.1-1.4

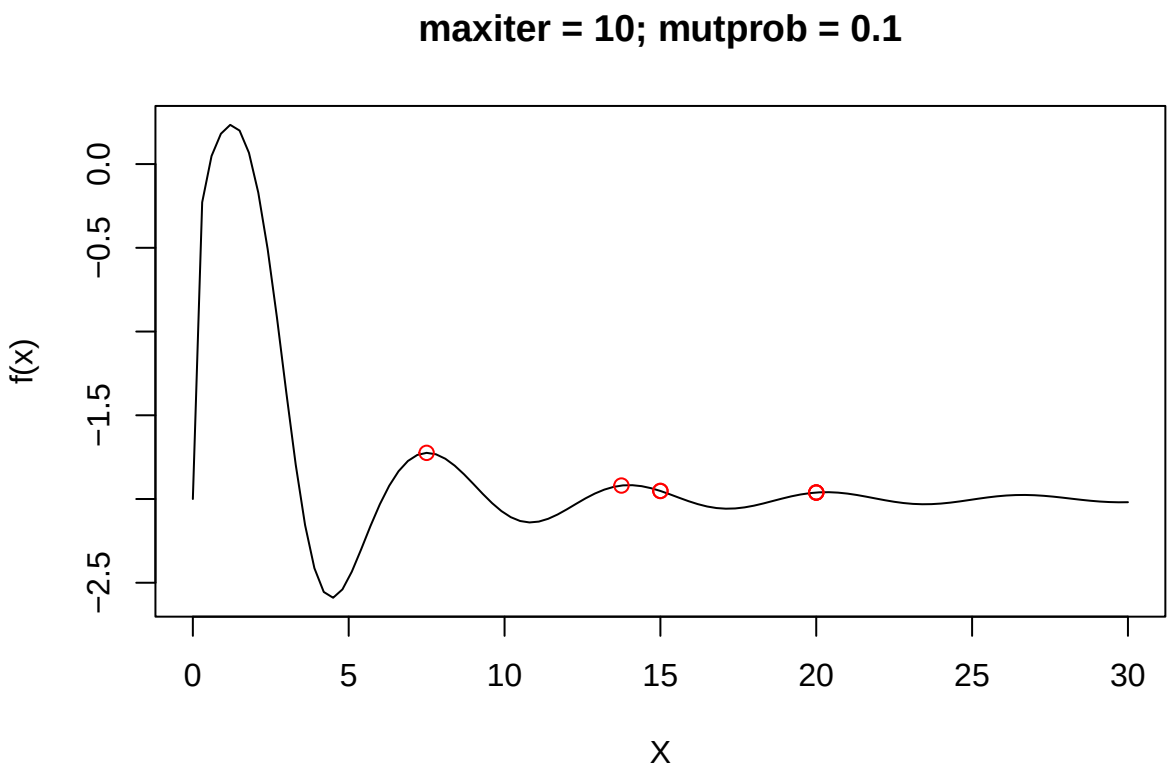
See codes below:

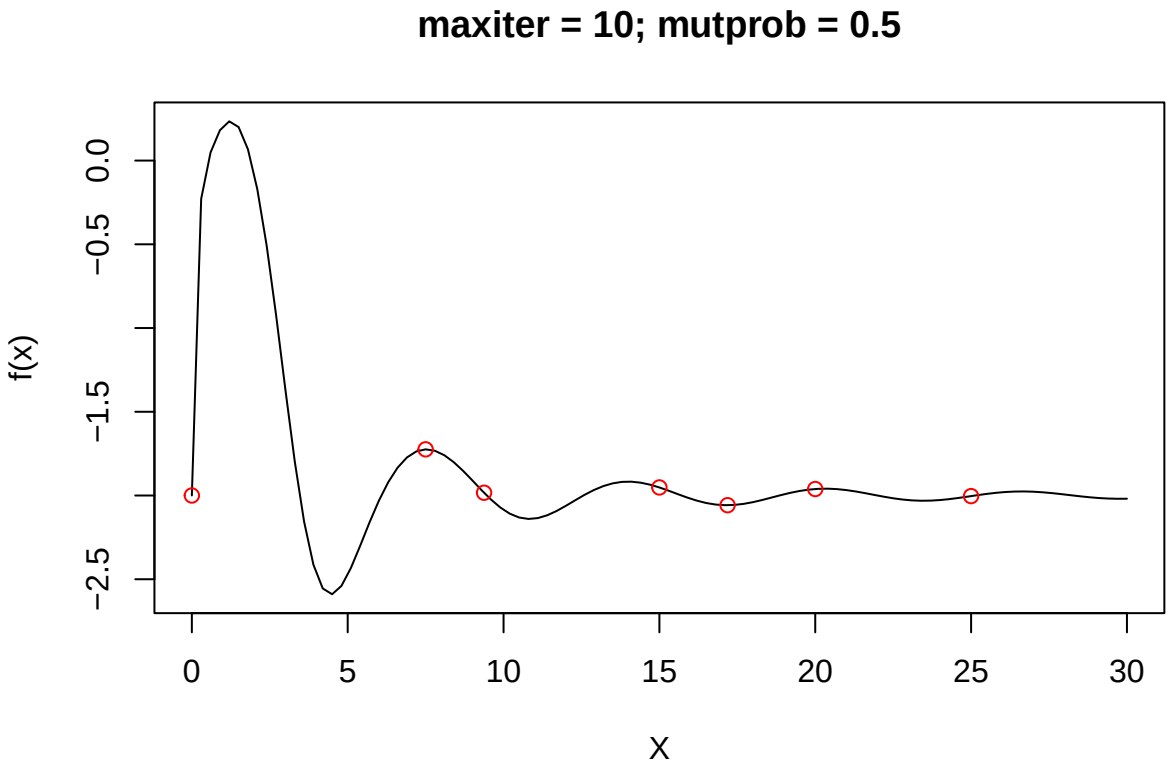
```
fxQ1 <- function(x){
  return(x^2 / exp(x) - 2*exp(-9*sin(x) / (x^2 + x + 1)))
}
# 1. Define the function

crossover <- function(x,y){
  return((x+y)/2)
}
# 2. Define the function crossover()

mutate <- function(x){
  return(x^2 %% 30)
}
# 3. Define the function mutate()

funQ14 <- function(maxiter, mutprob){
  curve(fxQ1, from = 0, to = 30, xlab = "X", ylab = "f(x)")
  title(main = paste0("maxiter = ",maxiter,"; ","mutprob = ", mutprob))
  # (a) Plots function f in the range from 0 to 30
  X <- seq(0,30,5)
  # (b) Defines an initial population

  Values <- fxQ1(X)
  # (c) Computes vector Values contains the function values
  maxf <- vector(length = maxiter)
  for (i in 1:maxiter) {
    # (d) Performs maxiter iterations
    indexP <- sample(1:length(X), 2)
    # i. Two indexes are randomly sampled from the current population, used as parents
    victim <- order(Values)[1]
    # ii. One index with the smallest objective function
    kid <- crossover(X[indexP[1]], X[indexP[2]])
    mutateFlag <- as.logical(sample(c(0,1), 1, prob = c(1-mutprob, mutprob)))
    if (mutateFlag) {
      kid <- mutate(kid)
    }
    # iii. Parents are used to produce a new kid by crossover,Mutate this kid with probability mutprob
```

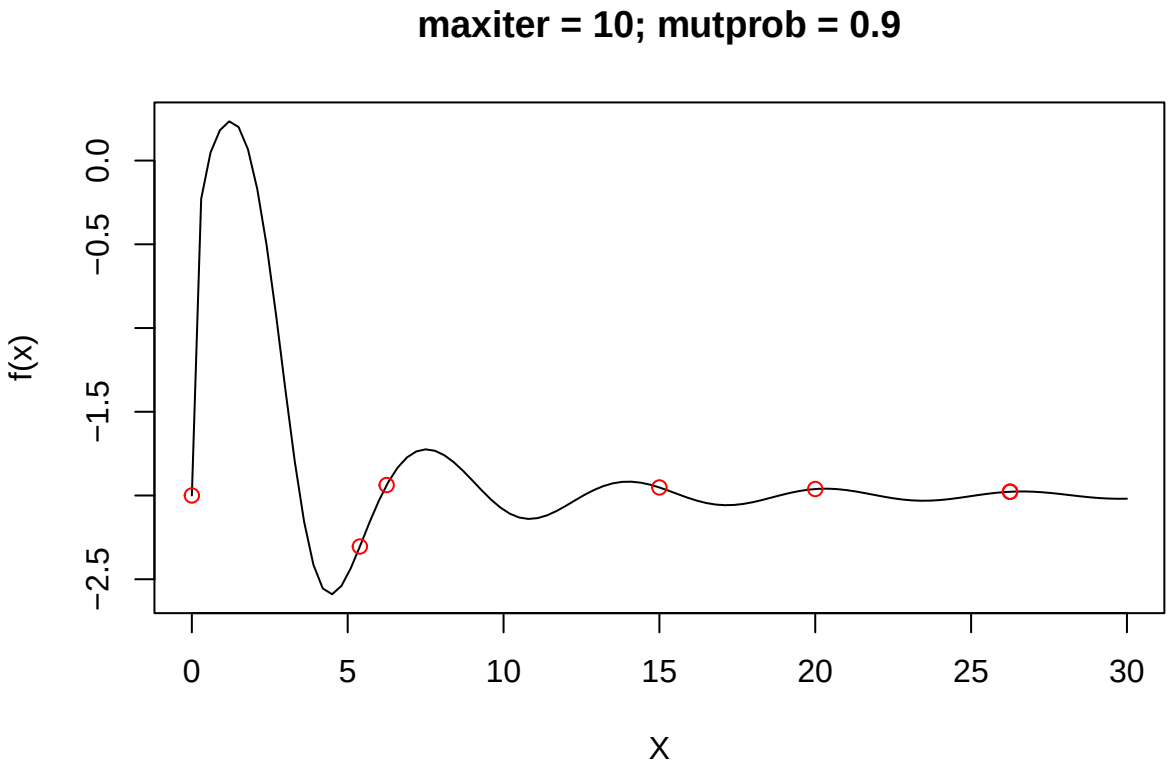
```
    X[victim] <- kid
    # iv. The victim is replaced by the kid in the population
    Values <- fxQ1(X)
    # and the vector Values is updated
    maxf[i] <- max(Values)
    # v. The current maximal value of the objective function is saved.
  }
    points(x = X, y = Values, col = "red")
    # (e) Add the final observations to the current plot in another colour.
    return(maxf)
}
```

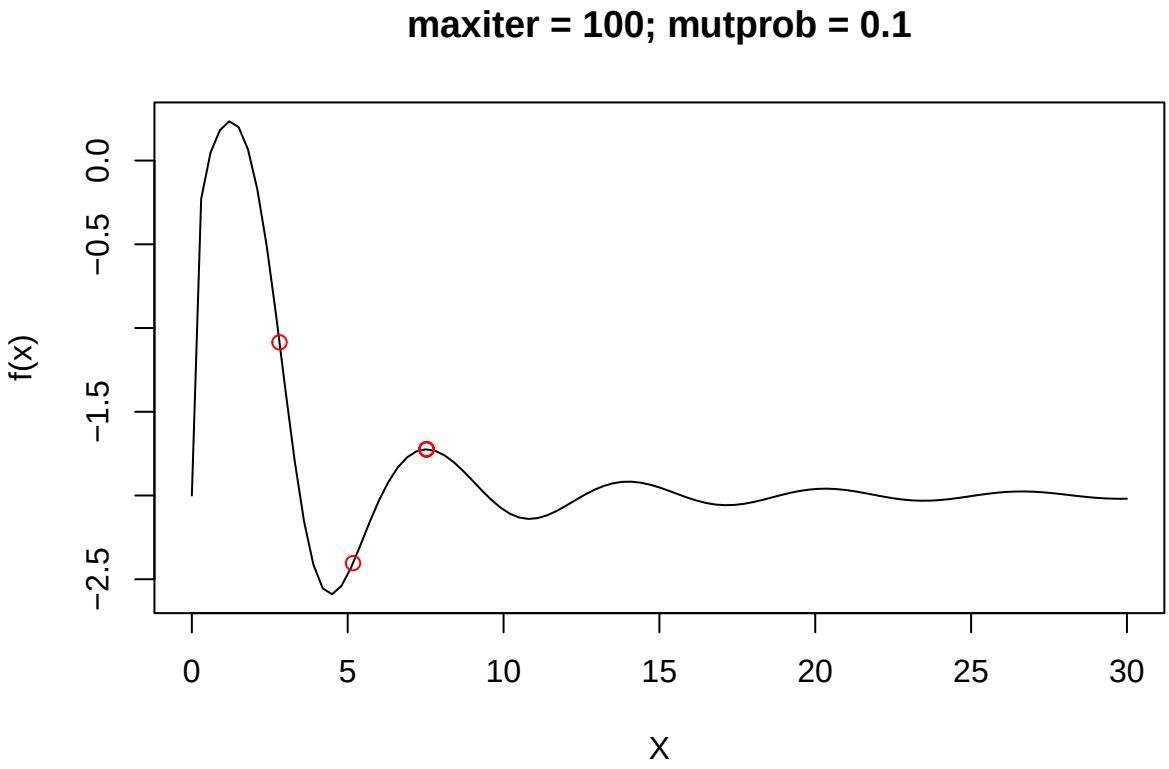
1.5

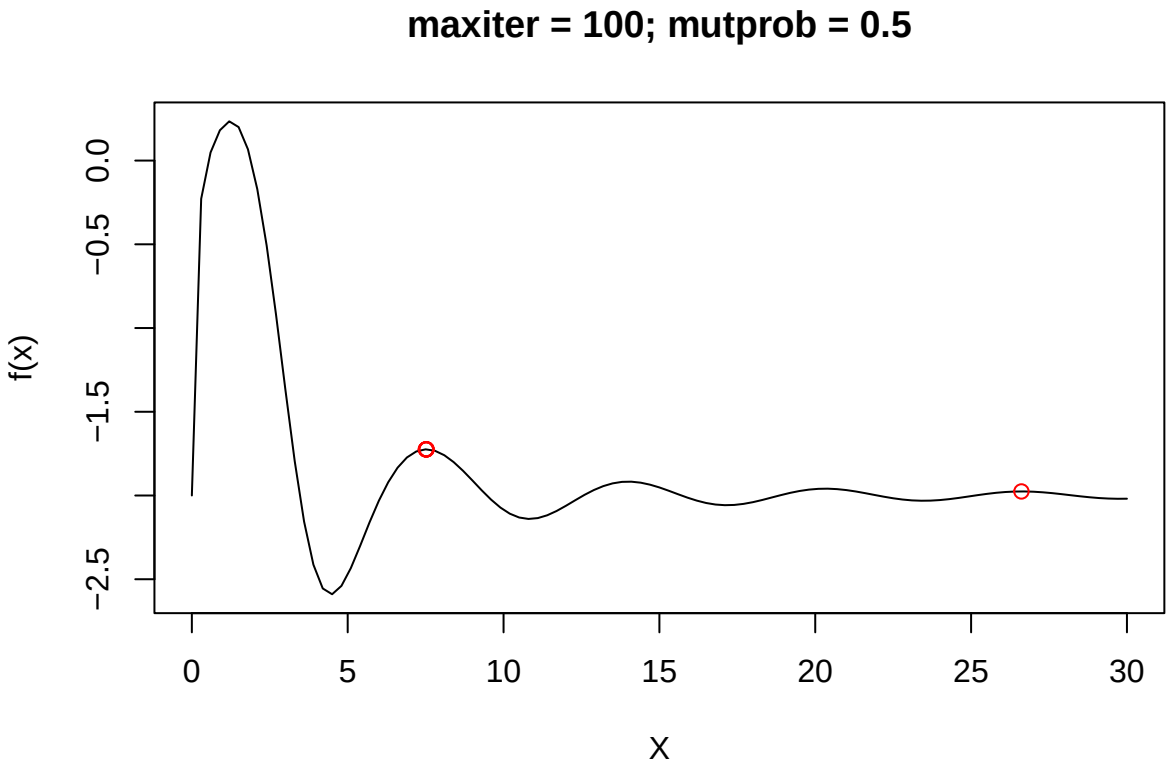
```
set.seed(98765)

maxFvalue <- array(dim = c(2,3,100))
Viter <- c(10,100)
Vprob <- c(0.1, 0.5, 0.9)
for (maxiter in Viter) {
  for (mutprob in Vprob) {
    maxFvalue[which(Viter == maxiter), which(Vprob == mutprob), ] <-
      funQ14(maxiter, mutprob)
  }
}
```

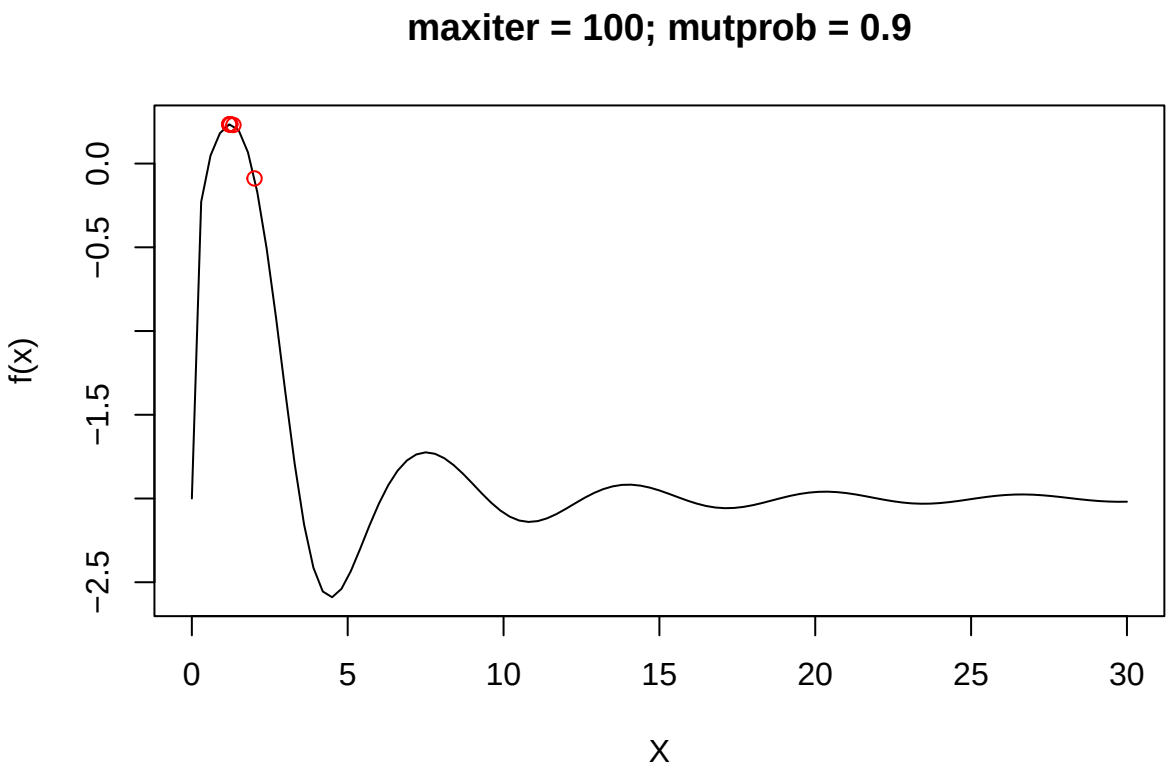



maxiter = 10; mutprob = 0.5
f(x)
0.0
−0.5
−1.5
−2.5
0
5
10
15
20
25
30
X


maxiter = 10; mutprob = 0.9
f(x)
0.0
−0.5
−1.5
−2.5
0
5
10
15
20
25
30
X


maxiter = 100; mutprob = 0.1
f(x)
0.0
−0.5
−1.5
−2.5
0
5
10
15
20
25
30
X


maxiter = 100; mutprob = 0.5
f(x)
0.0
−0.5
−1.5
−2.5
0
5
10
15
20
25
30
X




```
optimValue <- optim(1,fxQ1, method = "BFGS", control = list(fnscale = -1))
optimValue$value
```

```
## [1] 0.234853
```

```
maxFvalue[2,3,100]
```

```
## [1] 0.2348431
```

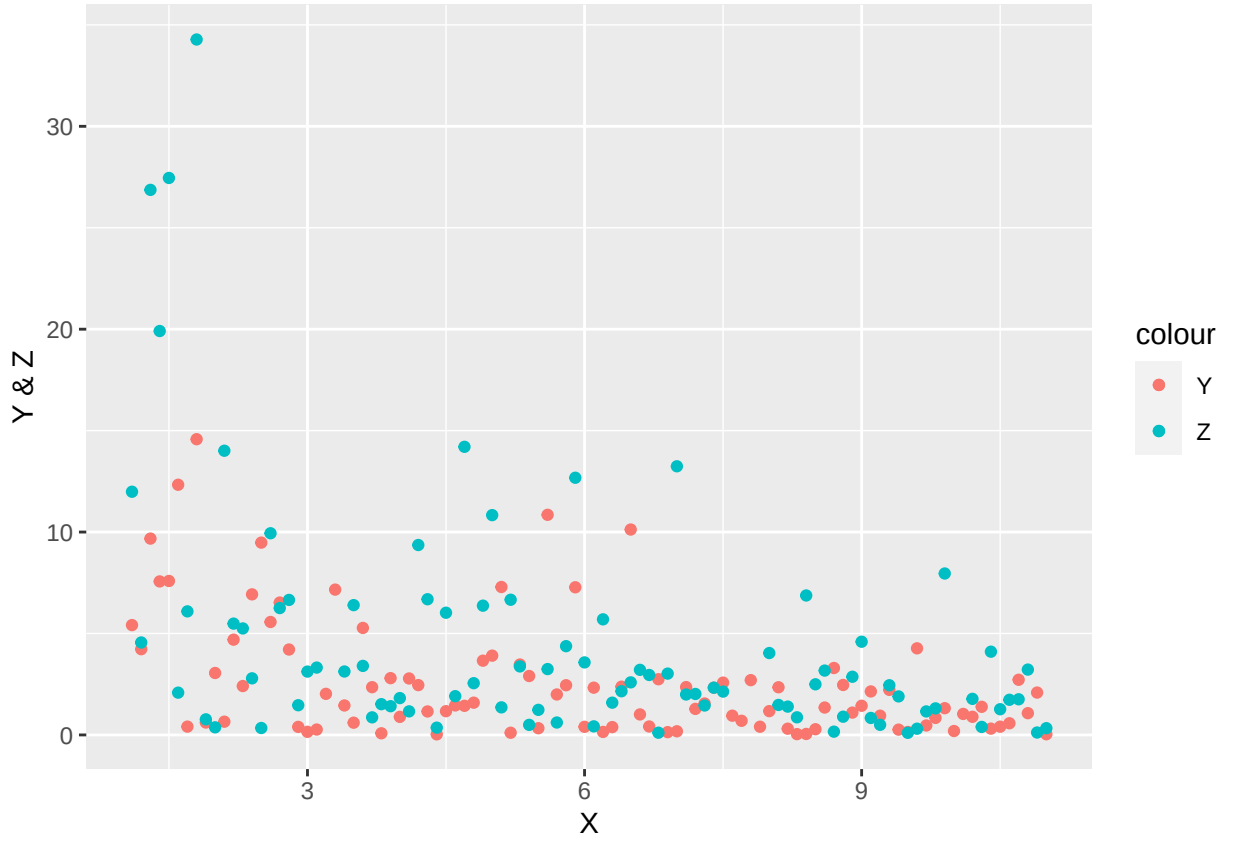
- From the plot we can see that when iteration = 10, and only 1 victim will be replaced each iteration, now enough iteration makes it can only do processes more like "reduce the variance" than "finding optimal values". It just choose X with lowest values and make it the mean of 2 others. And mutate cannot actually help because in most interval of this function, the value are on a similar level. We need to be very lucky to make the only few mutated value falls close to our global optimal.
  Then when iteration = 100, the iterations are enough to make our y-values close to each other, but still not yet enough to find the global optimal without enough mutations for the same reason that it may possibly falls into another low value(e.g. local maximum) again,which is far away from optimal value. When we choose iteration = 100 and mutation rate 0.9, finally we get a decent result.
  So the better choice (for accuracy) could be that use more iterations and relatively high mutation rate to make sure we can find our optimal value.

Question 2: EM algorithm

2.1

```
physical1 <- read.csv("physical1.csv")
```

```r
ggplot(data = physical1, aes(X,Y,Z))+
  geom_point(aes(x = X, y = Y, color = "Y"))+
  geom_point(aes(x = X, y = Z, color = "Z"))+
  ylab("Y & Z")
```



- Seems that Y and Z may possibly highly related to each other and sharing a same distribution depends on X. The response in general falls when X grows but always positive and seems converge to 0.

2.2

- See codes below

```r
FlambdaNew <- function(n, X, Y, Xz, Zobs, r, lambdak){
  1/(2*n)*(sum(X*Y) + 0.5*(sum(Xz*Zobs)) + (n-r)*lambdak)
}

funEM <- function(X, Y, Z, lambda0, minDelta, kmax){
  Zobs <- Z[!is.na(Z)]
  Zmiss <- Z[is.na(Z)]
  Xz <- X[which(!is.na(Z))]
  n <- length(c(Zobs, Zmiss))
  r <- length(Zobs)

  k <- 1
  lambdaPrev<-lambda0
  lambdaCurr<-lambdaPrev+100*minDelta
  print(c(lambda0, lambdaCurr))
```

```
  while ((abs(lambdaCurr-lambdaPrev)>minDelta) && (k<(kmax+1))){
    lambdaPrev<-lambdaCurr

    lambdaCurr <- FlambdaNew(n, X, Y, Xz, Zobs, r, lambdaPrev)


    k <- k+1
    cat("lambdaCurr:",lambdaCurr," ")
    cat("lambdaPrev:",lambdaPrev," ")
    cat("iteration:",k-1,"\n")
  }
}
```

2.3

```
X <- physical1$X
Y <- physical1$Y
Z <- physical1$Z

funEM(X,Y,Z,100 , 0.001, 100)
```

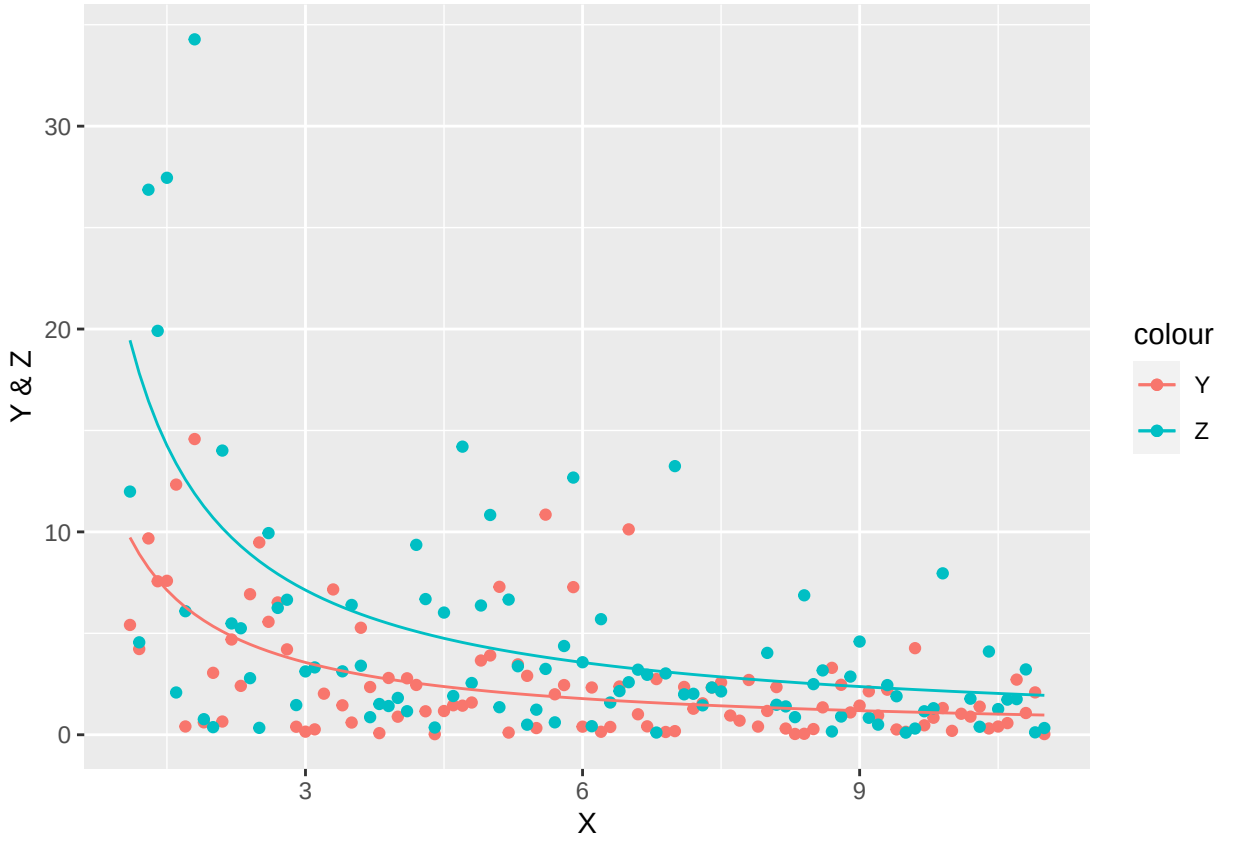
```
## [1] 100.0 100.1
## lambdaCurr: 14.27182  lambdaPrev: 100.1  iteration: 1
## lambdaCurr: 10.83869  lambdaPrev: 14.27182  iteration: 2
## lambdaCurr: 10.70137  lambdaPrev: 10.83869  iteration: 3
## lambdaCurr: 10.69588  lambdaPrev: 10.70137  iteration: 4
## lambdaCurr: 10.69566  lambdaPrev: 10.69588  iteration: 5
```

- The optimal value would be 10.69566, and it takes 5 iterations to reach it.

2.4

- Take 10.69566 as λ

```
optLambda <- 10.69566

EY <- optLambda / X
EZ <- 2*optLambda / (X)

ggplot(data = physical1)+
  geom_point(aes(x = X, y = Y, color = "Y"))+
  geom_point(aes(x = X, y = Z, color = "Z"))+
  geom_line(aes(x = X, y = EY, color = "Y"))+
  geom_line(aes(x = X, y = EZ, color = "Z"))+
  ylab("Y & Z")
```



- The λ we computed seems reasonable. Y value is generally lower than Z for the same X, as well as the expectation line we draw.

Appendix

```
library(ggplot2)


fxQ1 <- function(x){
  return(x^2 / exp(x) - 2*exp(-9*sin(x) / (x^2 + x + 1)))
}
# 1. Define the function

crossover <- function(x,y){
  return((x+y)/2)
}
# 2. Define the function crossover()

mutate <- function(x){
  return(x^2 %% 30)
}
# 3. Define the function mutate()

funQ14 <- function(maxiter, mutprob){
  curve(fxQ1, from = 0, to = 30, xlab = "X", ylab = "f(x)")
```

```
  title(main = paste0("maxiter = ",maxiter,"; ","mutprob = ", mutprob))
  # (a) Plots function f in the range from 0 to 30
  X <- seq(0,30,5)
  # (b) Defines an initial population

  Values <- fxQ1(X)
  # (c) Computes vector Values contains the function values
  maxf <- vector(length = maxiter)
  for (i in 1:maxiter) {
    # (d) Performs maxiter iterations
    indexP <- sample(1:length(X), 2)
    # i. Two indexes are randomly sampled from the current population, used as parents
    victim <- order(Values)[1]
    # ii. One index with the smallest objective function
    kid <- crossover(X[indexP[1]], X[indexP[2]])
    mutateFlag <- as.logical(sample(c(0,1), 1, prob = c(1-mutprob, mutprob)))
    if (mutateFlag) {
      kid <- mutate(kid)
    }
    # iii. Parents are used to produce a new kid by crossover,Mutate this kid with probability mutprob
    X[victim] <- kid
    # iv. The victim is replaced by the kid in the population
    Values <- fxQ1(X)
    # and the vector Values is updated
    maxf[i] <- max(Values)
    # v. The current maximal value of the objective function is saved.
    # print(X)
    # print(Values)
    # print(i)
    # print("###")
  }
    points(x = X, y = Values, col = "red")
    # (e) Add the final observations to the current plot in another colour.
    return(maxf)
}
set.seed(98765)

maxFvalue <- array(dim = c(2,3,100))
Viter <- c(10,100)
Vprob <- c(0.1, 0.5, 0.9)
for (maxiter in Viter) {
  for (mutprob in Vprob) {
    maxFvalue[which(Viter == maxiter), which(Vprob == mutprob), ] <-
      funQ14(maxiter, mutprob)
  }
}

optimValue <- optim(1,fxQ1, method = "BFGS", control = list(fnscale = -1))
optimValue$value
maxFvalue[2,3,100]


physical1 <- read.csv("physical1.csv")
```

```
ggplot(data = physical1, aes(X,Y,Z))+
  geom_point(aes(x = X, y = Y, color = "Y"))+
  geom_point(aes(x = X, y = Z, color = "Z"))+
  ylab("Y & Z")


FlambdaNew <- function(n, X, Y, Xz, Zobs, r, lambdak){
  1/(2*n)*(sum(X*Y) + 0.5*(sum(Xz*Zobs)) + (n-r)*lambdak)
}

funEM <- function(X, Y, Z, lambda0, minDelta, kmax){
  Zobs <- Z[!is.na(Z)]
  Zmiss <- Z[is.na(Z)]
  Xz <- X[which(!is.na(Z))]
  n <- length(c(Zobs, Zmiss))
  r <- length(Zobs)

  k <- 1
  lambdaPrev<-lambda0
  lambdaCurr<-lambdaPrev+100*minDelta
  print(c(lambda0, lambdaCurr))

  while ((abs(lambdaCurr-lambdaPrev)>minDelta) && (k<(kmax+1))){
    lambdaPrev<-lambdaCurr

    lambdaCurr <- FlambdaNew(n, X, Y, Xz, Zobs, r, lambdaPrev)


    k <- k+1
    cat("lambdaCurr:",lambdaCurr," ")
    cat("lambdaPrev:",lambdaPrev," ")
    cat("iteration:",k-1,"\n")
  }
}

X <- physical1$X
Y <- physical1$Y
Z <- physical1$Z

funEM(X,Y,Z,100 , 0.001, 100)


optLambda <- 10.69566

EY <- optLambda / X
EZ <- 2*optLambda / (X)

ggplot(data = physical1)+
  geom_point(aes(x = X, y = Y, color = "Y"))+
  geom_point(aes(x = X, y = Z, color = "Z"))+
  geom_line(aes(x = X, y = EY, color = "Y"))+
  geom_line(aes(x = X, y = EZ, color = "Z"))+
  ylab("Y & Z")
```