Computer Lab 6 Computational Statistics

Phillip Hölscher (phiho267) & Zijie Feng (zijfe244)

1 3 2019

Contents

Question 1: Genetic algorithm **2**
1. Define the function 2
2. Define the function `crossover()` 2
3. Define the function `mutate()` 2
4. Write a function 2
5. Run your code with different combinations 4

Question 2: EM algorithm **6**
1. Make a time series plot 6
2. Derive an EM algorithm that estimates λ 7
3.Implement this algorithm in R 8
4. Plot $E[Y]$ and $E[Z]$ versus X 9

Appendix **11**

Question 1: Genetic algorithm

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

1. Define the function

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

```
# define the function
func = function(x){
  return((x^2/exp(x)) - 2 * exp(-(9 * sin(x))/ (x^2 +x +1)))
}
```

2. Define the function `crossover()`

for two scalars x and y it returns their "kid as $(x + y)/2$.

```
# crossover function
crossover = function(x,y){
  kid = (x+y)/2
  return(kid)
}
```

3. Define the function `mutate()`

that for a scalar x returns the result of the integer division x2 mod 30. (Operation mod is denoted in R as %%).

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

4. Write a function

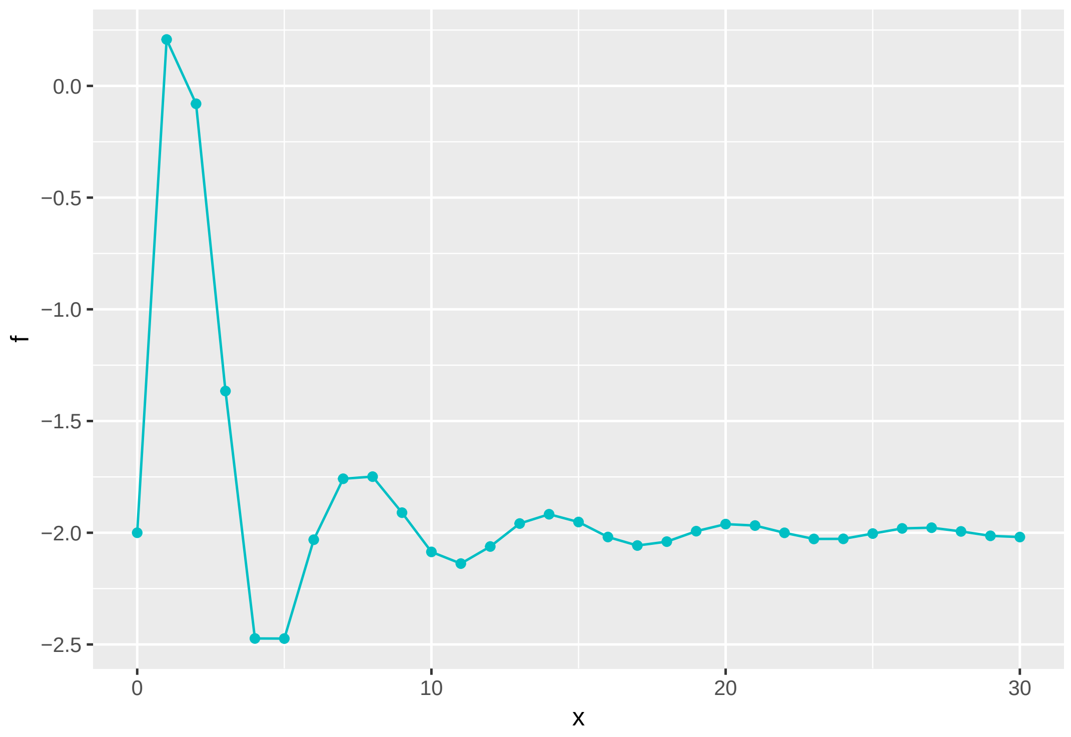
that depends on the parameters `maxiter` and `mutprob` and:

- (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
  - i. Two indexes are randomly sampled from the current population, they are further used as parents (use *sample()*).
  - ii. One index with the smallest objective function is selected from the current population, the point is referred to as victim (use *order()*).
  - iii. Parents are used to produce a new kid by crossover. Mutate this kid with probability *mutprob* (use *crossover()*, *mutate()*).

  - iv. The victim is replaced by the kid in the population and the vector *Values* is updated.
  - v. The current maximal value of the objective function is saved.

- (e) Add the final observations to the current plot in another colour.

```r
#1.4a##############################################################
dataa <- data.frame(x=0:30,f=func(0:30))
plot1.4=ggplot(dataa,aes(x=x,y=f))+
  geom_line(color="#00BFC4")+
  geom_point(color="#00BFC4")   # max x=1
plot1.4
```

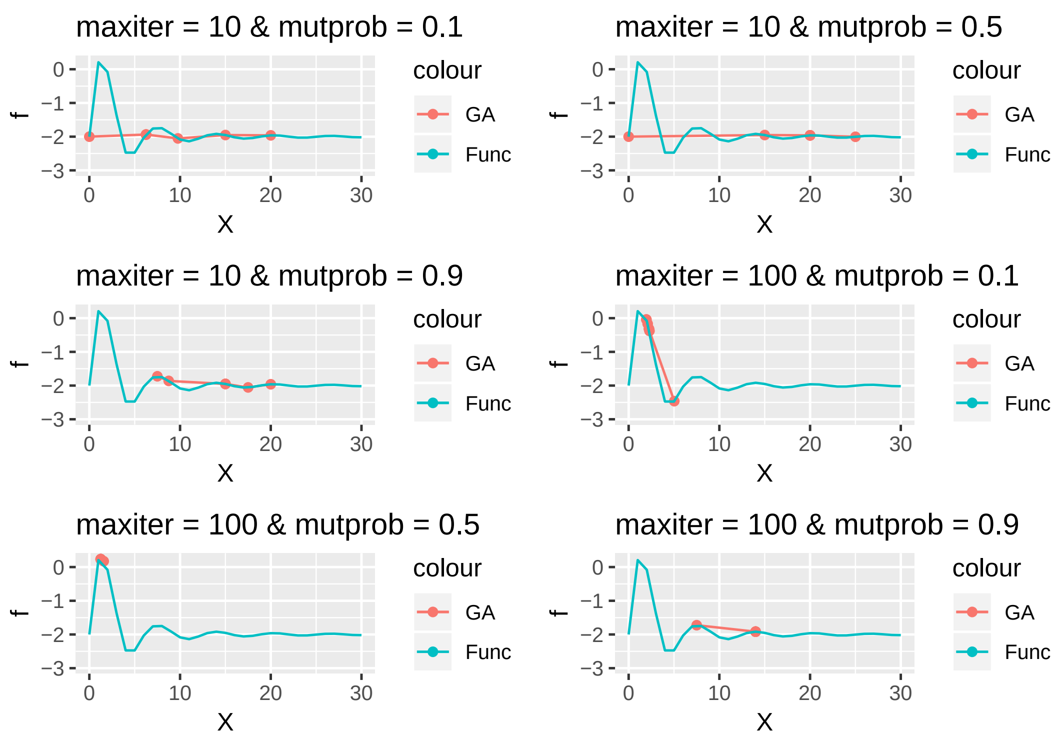


The maximum value might be between 0 and 2.

```r
#1.4b##############################################################
# initial population
X <- seq(0,30,5)

#1.4c##############################################################
# the function values for each population points
Values <- func(X)

#1.4d##############################################################
set.seed(1234567)
func4 <- function(pars, animation = F){
  maxiter = pars$maxiter
  mutprob = pars$mutprob
  name = pars$name
  tX <- X
  for(i in 1:maxiter) {
    samples <- sample(tX, 2, replace = F)
    id <- which.min(func(tX))
    kid <- crossover(samples[1],samples[2])
    if(runif(1)>mutprob){
      kid <- mutate(kid)
    }
    tX[id] <- kid
    tX <- sort(tX)
    if(animation){    # Here we can see the change step by step
      plot(tX,func(tX),type = "b",xlim=c(0,30), ylim = c(-3,0.25),col="Blue")
      lines(x=seq(0,30),y=f(seq(0,30)))
      Sys.sleep(0.2)
    }
  }

#1.4e##############################################################
  dt <- data.frame(X=tX,f=func(tX))
  pl = ggplot(dt,aes(x=X,y=f,color="Blue"))+
    geom_point()+
    geom_line()+
    geom_line(data=dataa,aes(x=x,y=f,color="Red"))+
    ylim(-3,0.25)+
    xlim(0,30)+
    ggtitle(name)+
    scale_color_discrete(labels=c("GA","Func"))
  return(pl)
}
```

5. 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?



To begin with, the results with maxiter=100 have better average performance than the ones with maxiter=10. Most of them get the maximum around X=1, except for the one with mutprob=0.9.

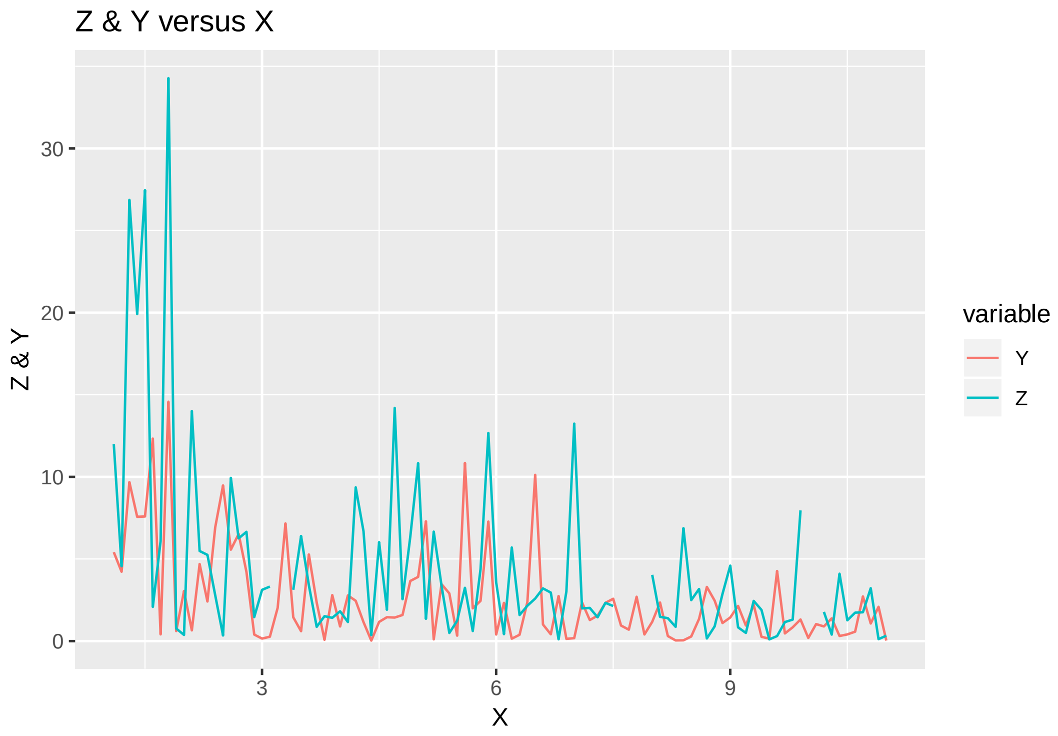
Based on our code, mutprob represents threshold of mutation. If it is large, the childern would be hard to mutate. According to the plots, large mutprob might restrict the search range, which help us find a local optimal, not a global optimal.

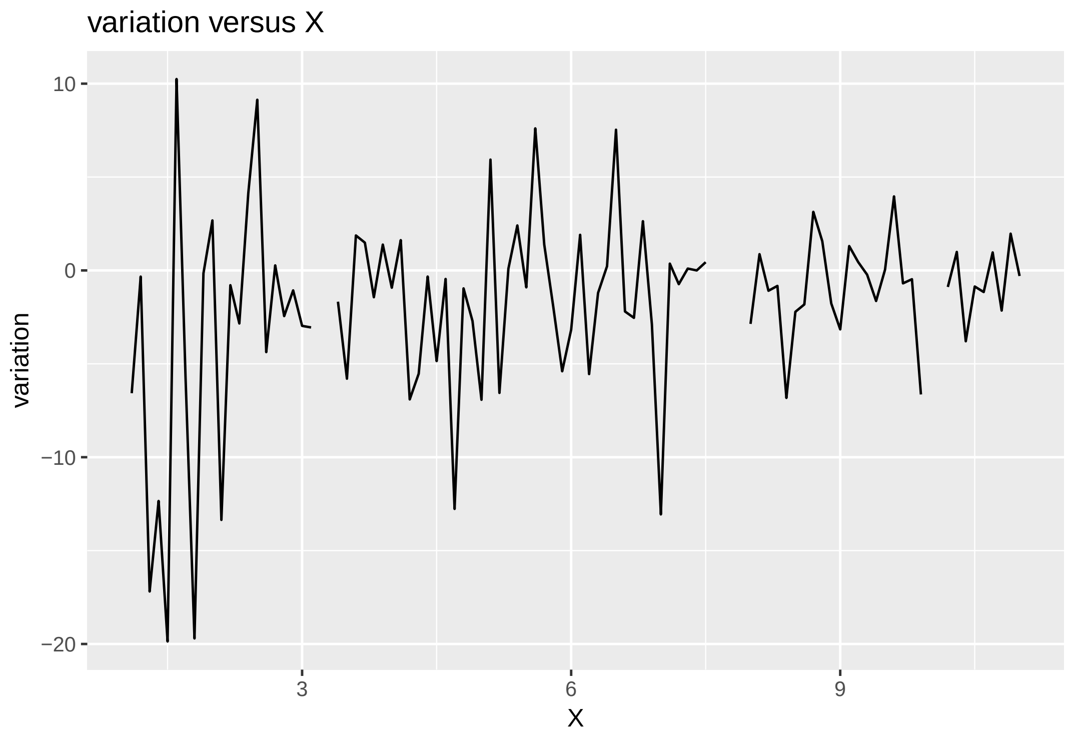
Question 2: EM algorithm

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

1. 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?





It does not seem that the two processes are related to each other. In the beginning, the Z value does sleep off more than double of Y. Also, the movements of the processes rarely lie on top of each other. Based on the variation of such two responses, we can recognize the Z values are incomplete in some parts, the curve has gaps. However, the variation seems to decrease when X increases.

2. Derive an EM algorithm that estimates λ

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

$$Y_i \sim \exp(X_i/\lambda), \quad Z_i \sim \exp(X_i/2\lambda)$$

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

EM algorithm is a algorithm based on maximum likelihood estimation. We have to find the a probability density function (PDF) $p(x, z, \theta)$, where x is a sequence of observed data , z is latent data and θ are unknown parameters. Afterwards, there are two steps. Firstly, we expose the latent data z by maximum log-liklihood and current data x. Secondly, we get the estimate of parameters θ based on the maximum log-liklihood.

Expectation

In our case, we know the PDFs of Y and Z are

$$f(Y_i) = \frac{X_i}{\lambda} e^{-\frac{X_i}{\lambda} Y_i}, f(Z_i) = \frac{X_i}{2\lambda} e^{-\frac{X_i}{2\lambda} Z_i},$$

where the missing values in Z can be considered as latent data and the others are observed data. Since we consider that all the data are independent and we can thereby get the liklihood, which is

$$\mathcal{L}(\lambda|Y,Z) = \prod f(Y_i) \cdot \prod f(Z_i)$$

$$= \prod(\frac{X_i}{\lambda} e^{-\frac{X_i}{\lambda}Y_i}) \ \cdot \ \prod(\frac{X_i}{2\lambda} e^{-\frac{X_i}{2\lambda}Z_i})$$

$$= \frac{\prod X_i}{\lambda^n} e^{-\frac{\sum X_i Y_i}{\lambda}} \ \cdot \ \frac{\prod X_i}{2^n \lambda^n} e^{-\frac{\sum X_i Z_i}{2\lambda}} \ .$$

Thus, the log-likelihood should be

$$\log \mathcal{L}(\lambda|Y,Z) = \prod(\ln X_i) - n\ln\lambda - \frac{\sum X_i Y_i}{\lambda} + \prod(\ln X_i) - n\ln(2\lambda) - \frac{1}{2\lambda}\sum X_i Z_i.$$

In the expectation step, we use maximum log-liklihood estimate to expose/express the missing data Z_{miss} by the other data (Y, Z_{obs}) in the last iteration, and thereby estimate our target paramter λ. Additionally, we replace the miss data with the expected value of Z.

$$Q(\lambda,\lambda_k) = E[\log \mathcal{L}(\lambda|Y, Z_{miss})|(\lambda_k, Y, Z_{obs})]$$

$$= \prod(\ln X_i) - n\ln\lambda_k - \frac{\sum X_i Y_i}{\lambda_k} + \prod(\ln X_i) - n\ln(2\lambda_k) - \frac{1}{2\lambda_k}[\sum_{obs} X_i Z_i + \sum_{miss} X_i \frac{2\lambda_{k-1}}{X_i}]$$

$$= 2\prod(\ln X_i) - n\ln\lambda_k - \frac{\sum X_i Y_i}{\lambda_k} - n\ln(2\lambda_k) - \frac{1}{2\lambda_k}\sum_{obs} X_i Z_i - \frac{1}{\lambda_k}\sum_{miss}\lambda_{k-1}.$$

maximization

To obtain estimate λ, we need to solve that the partial erivative of $Q(\lambda,\lambda_k)$ equals 0.

$$\nabla\lambda = -\frac{2n}{\lambda_k} + \frac{\sum X_i Y_i}{\lambda_k^2} + \frac{1}{2\lambda_k^2}\sum_{obs} X_i Z_i + \frac{1}{\lambda_k^2}\sum_{miss}\lambda_{k-1} = 0$$

$$\lambda_k = \frac{1}{4n}(2\sum X_i Y_i + \sum_{obs} X_i Z_i + 2\sum_{miss}\lambda_{k-1})$$

3.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?

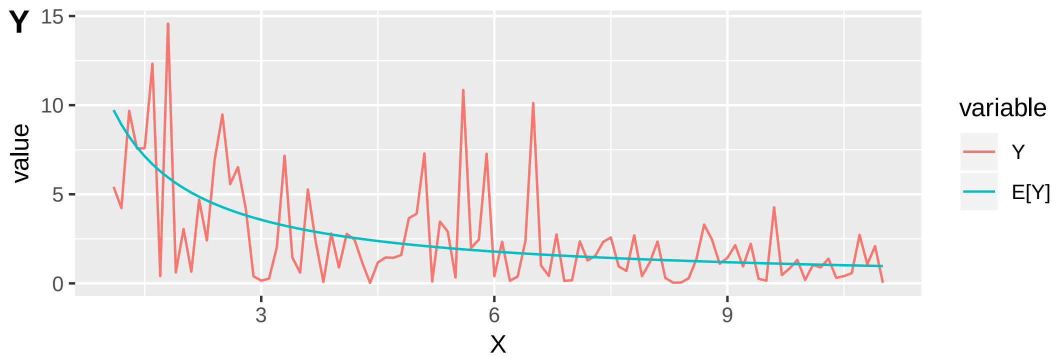
```
EM <- function(data2, eps, kmax=500, lda){
  X <- data2$X
  Y <- data2$Y
  Z <- data2$Z
  obs <- !is.na(Z)
  n <- length(X)
  m <- sum(is.na(Z))
  # browser()
  loglik <- function(lda,ldap){
    return(2*sum(log(X))-n*log(lda)-(X%*%Y)/lda-n*log(2*lda)
           -(X[obs]%*%Z[obs])/(2*lda)-m*ldap/lda)
  }
  # initial state
  k <- 0
  llvalprev <- lda*10+10  # make some difference
  llvalcurr <- lda
  llk <- loglik(llvalcurr,llvalprev)

  print(c("iter","lda","llk"))
  print(c(k, llvalcurr,llk))

  while ((abs(llvalprev-llvalcurr)>eps) && (k<(kmax+1))){
    llvalprev <- llvalcurr
    # since we already know how to get argmax lda,
    # we can use the solution directly
    llvalcurr <- 1/(4*n)*(2*X%*%Y+X[obs]%*%Z[obs]+2*m*llvalprev)

    k<-k+1
    llk <- loglik(llvalcurr,llvalprev)
    print(c(k, llvalcurr,llk))
  }
  return(llvalcurr)
}

lda_opt <- EM(data2, eps=0.001, lda=100)
```

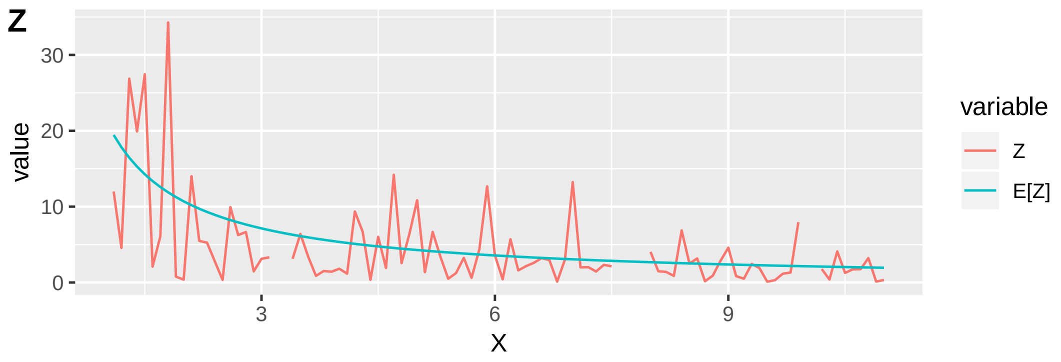
```
## [1] "iter" "lda"  "llk"
## [1]    0.0000  100.0000 -761.7647
## [1]    1.00000   14.26782 -470.99635
## [1]    2.00000   10.83853 -416.01655
## [1]    3.00000   10.70136 -413.46921
## [1]    4.00000   10.69587 -413.36664
## [1]    5.00000   10.69566 -413.36253
```

According to the code, it is obvious that the log-liklihood increases during iterations. We get the optimal $\lambda = 10.69566$ after 5 iterations.

4. 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.





In accordance with the plot, the paths (blue) of bath E[Y] and E[Z] pass through the original data (red), which show similar tendencies. We can therefore conclude that the λ we get is reasonable.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE, out.height = "280px")
# library used in this lab
library(ggplot2) # ex 2.1 - time series plot
library(gridExtra)
# clean the environment
rm(list=ls())
# define the function
func = function(x){
  return((x^2/exp(x)) - 2 * exp(-(9 * sin(x))/ (x^2 +x +1)))
}
# crossover function
crossover = function(x,y){
  kid = (x+y)/2
  return(kid)
}
# mutate function
mutate = function(x){
  return(x^2 %%30)
}
#1.4a##############################################################
dataa <- data.frame(x=0:30,f=func(0:30))
plot1.4=ggplot(dataa,aes(x=x,y=f))+
  geom_line(color="#00BFC4")+
  geom_point(color="#00BFC4")   # max x=1
plot1.4
#1.4b##############################################################
# initial population
X <- seq(0,30,5)

#1.4c##############################################################
# the function values for each population points
Values <- func(X)

#1.4d##############################################################
set.seed(1234567)
func4 <- function(pars, animation = F){
  maxiter = pars$maxiter
  mutprob = pars$mutprob
  name = pars$name
  tX <- X
  for(i in 1:maxiter) {
    samples <- sample(tX, 2, replace = F)
    id <- which.min(func(tX))
    kid <- crossover(samples[1],samples[2])
    if(runif(1)>mutprob){
      kid <- mutate(kid)
    }
    tX[id] <- kid
    tX <- sort(tX)
    if(animation){    # Here we can see the change step by step
      plot(tX,func(tX),type = "b",xlim=c(0,30), ylim = c(-3,0.25),col="Blue")
      lines(x=seq(0,30),y=f(seq(0,30)))
      Sys.sleep(0.2)
    }
  }

#1.4e################################################################
  dt <- data.frame(X=tX,f=func(tX))
  pl = ggplot(dt,aes(x=X,y=f,color="Blue"))+
    geom_point()+
    geom_line()+
    geom_line(data=dataa,aes(x=x,y=f,color="Red"))+
    ylim(-3,0.25)+
    xlim(0,30)+
    ggtitle(name)+
    scale_color_discrete(labels=c("GA","Func"))
  return(pl)
}
maxiter = c(10,100)
mutprob = c(0.1,0.5,0.9)
names = c("maxiter = 10 & mutprob = 0.1",
          "maxiter = 100 & mutprob = 0.5",
          "maxiter = 10 & mutprob = 0.9",
          "maxiter = 100 & mutprob = 0.1",
          "maxiter = 10 & mutprob = 0.5",
          "maxiter = 100 & mutprob = 0.9")
pairs = data.frame(maxiter=rep(maxiter,3),mutprob=rep(mutprob,2),name=names)
pairs = split(pairs,pairs[,3])

plot(arrangeGrob(grobs=lapply(t(pairs), func4)))
# clean the environment
rm(list=ls())
# load the data
data2 = read.csv("physical1.csv")
# Z & Y versus X

data21 <- reshape2::melt(data2, id.va = "X")
ggplot(data = data21,aes(x=X,y=value,color=variable)) +
  geom_line()+
  # geom_line(aes(x = X, y=Y),color = "#F8766D") +
  # geom_line(aes(x = X, y=Z),color = "#00BFC4") +
  ggtitle("Z & Y versus X") +
  ylab("Z & Y")

ggplot(data = data.frame(X=data2$X,variation=data2$Y-data2$Z), aes(x = X)) +
  geom_line(aes(y = variation)) +
  ggtitle("variation versus X")
EM <- function(data2, eps, kmax=500, lda){
  X <- data2$X
  Y <- data2$Y
  Z <- data2$Z
  obs <- !is.na(Z)
  n <- length(X)
  m <- sum(is.na(Z))
```

```r
  # browser()
  loglik <- function(lda,ldap){
    return(2*sum(log(X))-n*log(lda)-(X%*%Y)/lda-n*log(2*lda)
           -(X[obs]%*%Z[obs])/(2*lda)-m*ldap/lda)
  }
  # initial state
  k <- 0
  llvalprev <- lda*10+10  # make some difference
  llvalcurr <- lda
  llk <- loglik(llvalcurr,llvalprev)

  print(c("iter","lda","llk"))
  print(c(k, llvalcurr,llk))

  while ((abs(llvalprev-llvalcurr)>eps) && (k<(kmax+1))){
    llvalprev <- llvalcurr
    # since we already know how to get argmax lda,
    # we can use the solution directly
    llvalcurr <- 1/(4*n)*(2*X%*%Y+X[obs]%*%Z[obs]+2*m*llvalprev)

    k<-k+1
    llk <- loglik(llvalcurr,llvalprev)
    print(c(k, llvalcurr,llk))
  }
  return(llvalcurr)
}

lda_opt <- EM(data2, eps=0.001, lda=100)
X <- data2$X
EY <- c(lda_opt)/X
EZ <- 2*EY

data241 <- cbind(data2[,-3],data.frame(EY=EY))
data241 <- reshape2::melt(data241, id.va="X")
p1 <- ggplot(data241, aes(x=X, y=value, color=variable))+
  geom_line()+
  scale_color_discrete(labels=c("Y","E[Y]"))

data242 <- cbind(data2[,-2],data.frame(EY=EZ))
data242 <- reshape2::melt(data242, id.va="X")
p2 <- ggplot(data242, aes(x=X, y=value, color=variable))+
  geom_line()+
  scale_color_discrete(labels=c("Z","E[Z]"))

cowplot::plot_grid(p1, p2, labels = c('Y', 'Z'),ncol=1)
```