STAT 243 PS 8

Junyuan Gao(SID:26484653)

December 1, 2017

1 Q1

1.1 a

The density of exponential random variable X is

$$f(x) = \lambda e^{-\lambda x}$$

In this question we need to compare the first derivatives of f(x) and p(x).
As

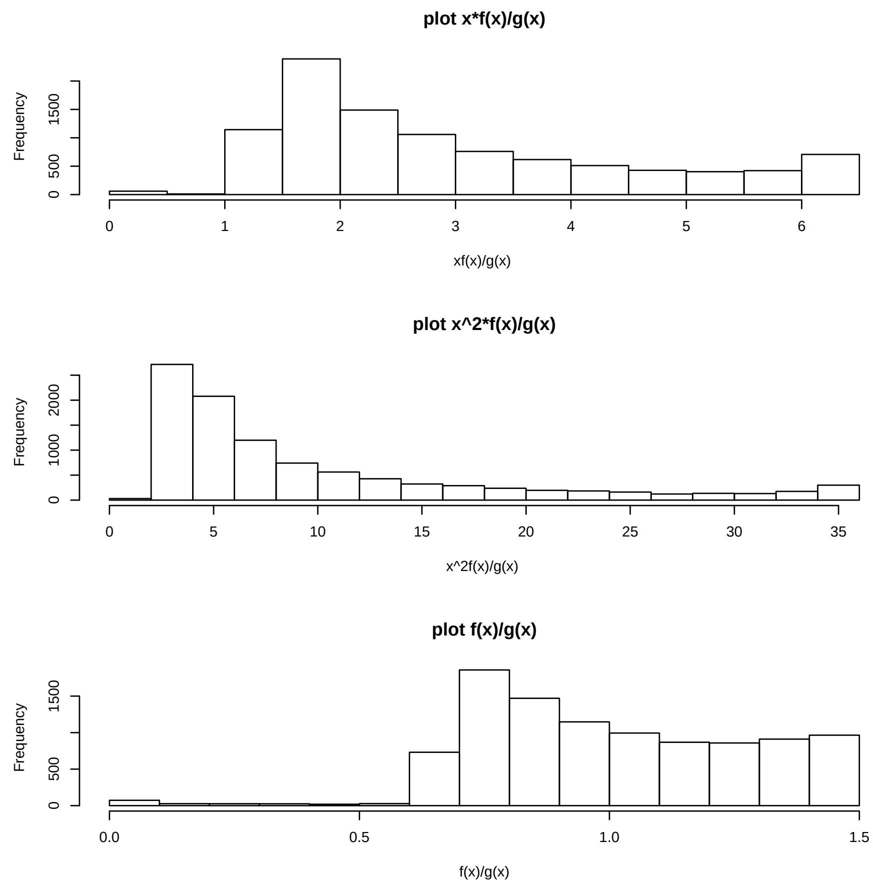
$$\lim_{x\to\infty} \frac{f'(x)}{p'(x)} = \lim_{x\to\infty} \frac{\lambda^2 x^{\beta+2} e^{-\lambda x}}{\beta\alpha^\beta(\beta+1)} = \lim_{x\to\infty} \frac{\lambda^2 x^{\beta+2}}{\beta\alpha^\beta(\beta+1)e^{\lambda x}} = 0$$

by L-Hospital Rule. Thus, the Pareto decay is more slowly than that of an exponential distribution.

1.2 b

```
library(EnvStats)

## Warning:  package 'EnvStats' was built under R version 3.4.2

#Generate 10000 samps from pareto dfistribution
samp<-rpareto(10000, location = 2, shape = 3)
#estimate EX
EX<-samp*dexp(samp-2, rate = 1, log = FALSE)/dpareto(samp,
                    location = 2, shape =3)
mean(EX)

## [1] 2.994894

#estimate EX^2
EX2<-(samp^2)*dexp(samp-2, rate = 1, log = FALSE)/dpareto(samp,
                    location = 2, shape =3)
mean(EX2)
```

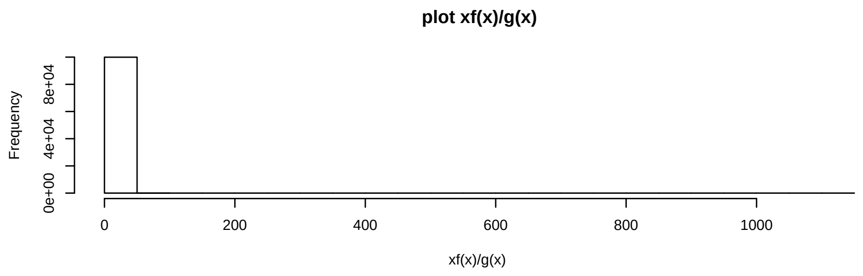
```
## [1] 9.96265

par(mfrow=c(3,1))
# histgram of h(x)f(x)/g(x), where h(x)=x
hist(EX, main= 'plot x*f(x)/g(x)', xlab = 'xf(x)/g(x) ')
# histgram of h(x)f(x)/g(x), where h(x)=x^2
hist(EX2, main= 'plot x^2*f(x)/g(x)', xlab = 'x^2f(x)/g(x) ')
# histgram of f(x)/g(x)
fg= dexp(samp-2, rate = 1, log = FALSE)/dpareto(samp,
                    location = 2, shape =3)
hist(fg, main = 'plot f(x)/g(x)', xlab = 'f(x)/g(x)')
```

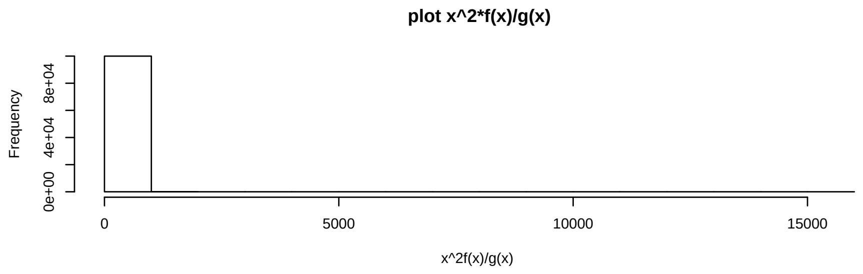


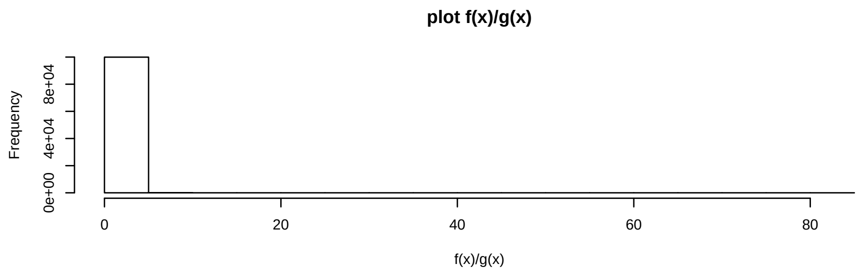
From the histograms, we can conclude that the range of weights $\frac{f(x)}{g(x)}$ is from 0 to 1.5, which is a very small range and implies that there are no extreme weights.

1.3 c

```
#Generate 10000 samples from pareto dfistribution
sampp<-rexp(100000, rate = 1)+2
#estimate EX
EX<-sampp*dpareto(sampp, location = 2,
                  shape =3)/dexp(sampp-2, rate = 1, log = FALSE)
mean(EX)

## [1] 2.928054

#estimate EX^2
EX2<-(sampp^2)*dpareto(sampp, location = 2,
                       shape =3)/dexp(sampp-2, rate = 1, log = FALSE)
mean(EX2)

## [1] 10.19347

# histgram of h(x)f(x)/g(x), where h(x)=x
par(mfrow=c(3,1))
hist(EX,main = 'plot xf(x)/g(x)', xlab = 'xf(x)/g(x) ')
# histgram of h(x)f(x)/g(x), where h(x)=x^2
hist(EX2,main = 'plot x^2*f(x)/g(x)', xlab = 'x^2f(x)/g(x) ')
# histgram of f(x)/g(x)
ratio <-dpareto(sampp, location = 2, shape =3)/dexp(sampp-2,
                                 rate = 1, log = FALSE)
hist(ratio, main = 'plot f(x)/g(x)', xlab = 'f(x)/g(x)')
```







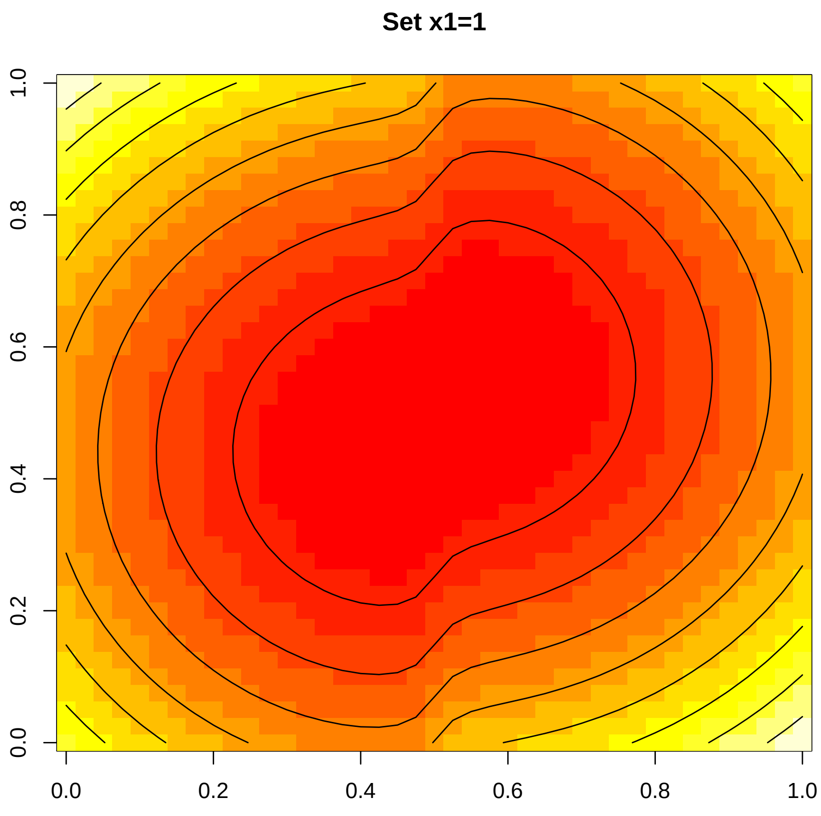
From the two histograms of $\frac{xf(x)}{g(x)}$ and $\frac{x^2f(x)}{g(x)}$, most values are located in 1-10 of $\frac{h(x)f(x)}{g(x)}$. However, for the function $\frac{x^2f(x)}{g(x)}$, there are some very large values that makes the distribution long tail.

2 Q2

```
## problem 2
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)

f <- function(x) {
  f1 <- 10*(x[3] - 10*theta(x[1],x[2]))
  f2 <- 10*(sqrt(x[1]^2 + x[2]^2) - 1)
  f3 <- x[3]
  return(f1^2 + f2^2 + f3^2)
}
```

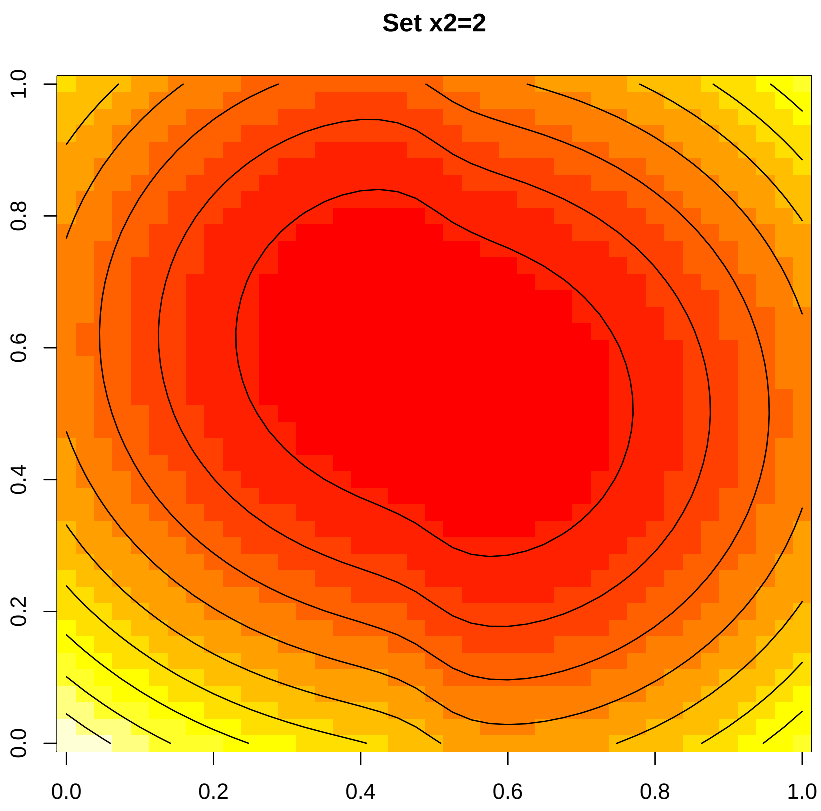
First fix $x_1 = 1$.

```
# plot f() fixed x[1]
x1 = 1
x2 = c(-20:20)
x3 = c(-20:20)
fx= data.frame()
for (i in 1:41){
  for (j in 1:41){
    fx[i,j]= f(c(x1, x2[i], x3[j]))
  }
}
image(as.matrix(fx), main= "Set x1=1")
contour(z=as.matrix(fx), add = TRUE,
        drawlabels = FALSE)
```



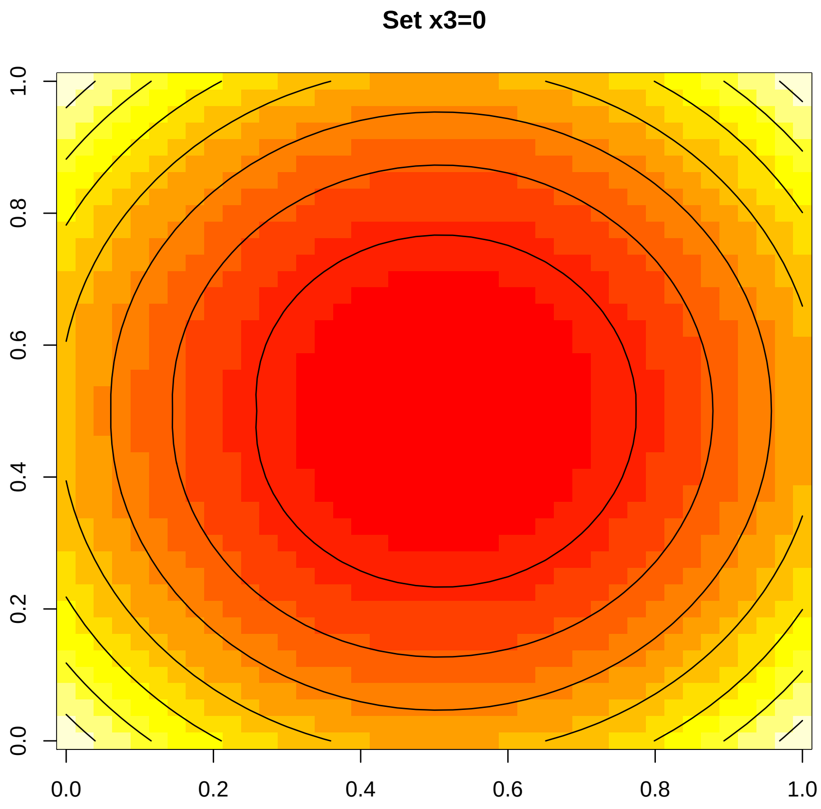
Then fix $x_2 = 2$

```r
x1 = c(-20:20)
x2 = 2
x3 = c(-20:20)
fx= data.frame()
for (i in 1:41){
  for (j in 1:41){
    fx[i,j]= f(c(x1[i], x2, x3[j]))
  }
}
image(as.matrix(fx), main= "Set x2=2")
contour(z=as.matrix(fx), add = TRUE,
        drawlabels = FALSE)
```



Then fix $x_3 = 0$

```
x1 = c(-20:20)
x2 = c(-20:20)
x3 = 0
fx= data.frame()
for (i in 1:41){
  for (j in 1:41){
    fx[i,j]= f(c(x1[i], x2[j], x3))
  }
}
image(as.matrix(fx), main= "Set x3=0")
contour(z=as.matrix(fx), add = TRUE,
        drawlabels = FALSE)
```



At last use optim() and nlm() to find the minimum of function.

```
#try different starting points
optim(c(1,1,1), f)$par

## [1]  0.9999779414 -0.0001349269 -0.0001927127

optim(c(1,1,1), f)$value

## [1] 1.343098e-07

nlm(f, c(1,1,1))$minimum

## [1] 1.702065e-08

nlm(f, c(1,1,1))$estimate

## [1]  9.999995e-01 -8.225859e-05 -1.301257e-04

######## first approach ###############
(rand_para= runif(3, -50, 50))

## [1]  45.350394   9.856512 -39.387206

optim(rand_para, f)$par

## [1] 1.003374738 0.003751077 0.006647949

optim(rand_para, f)$value

## [1] 0.001236543

nlm(f, rand_para)$minimum

## [1] 8.822081e-18

nlm(f, rand_para)$estimate

## [1] 1.000000e+00 9.798858e-10 1.766246e-09

######## second approach ###############
(rand_para= runif(3, -10, 10))

## [1] -0.48596750 -0.02733021  0.65460104

optim(rand_para, f)$par

## [1]  0.9996095625 -0.0003450278 -0.0006688660

optim(rand_para, f)$value

## [1] 1.711545e-05
```

```
nlm(f, rand_para)$minimum

## [1] 2.382387e-18

nlm(f, rand_para)$estimate

## [1] 1.000000e+00 7.113157e-10 1.216901e-09

######## third approach ###############
(rand_para= runif(3, -5, 5))

## [1] -1.9856944 -3.9571927 -0.6236504

optim(rand_para, f)$var

## NULL

optim(rand_para, f)$value

## [1] 1.516704e-05

nlm(f, rand_para)$minimum

## [1] 7.746659e-17

nlm(f, rand_para)$estimate

## [1]  1.000000e+00 -2.131589e-10  5.369104e-10
```

From above approaches, we can see that if starting with different starting points, the function will converge to same minima($(x_1, x_2, x_3) = (1, 0, 0)$). Thus, I think (1,0,0) will be the global minima and there may not be multiple local minimas.

3 Q3

3.1 a

In E step, we have $Q(\theta|\theta_t) = E_z[logL(\theta|Y)|X, \theta_t]$ and we denote Y=(X,Z) where Y is complete data, X is available data, and Z is censored data. Detailed deduction attached in **appendix paper**.

From the above paragraph and deduction in appendix, we have the following EM algorithm:
1.**E step**: compute

$$Q(\theta|\theta_t) = -nlog(\sigma) - \frac{n}{2}log2\pi - \frac{1}{2\sigma^2}[\sum_{i=1}^{c}((E(m_{i,t}) - \mu_{i,t})^2 + var(m_{i,t})) + \sum_{i=c+1}^{n}(Y_i - \mu_i)^2]$$

2.**M step**: based on the calculation in M step, calculate

$$\sigma^2_{t+1} = \frac{1}{n}(\sum_{i=1}^{n}(y_i^* - \beta_{0,t} - \beta_{1,t}x_i)^2 + \sum_{i=1}^{c} var(m_{i,t}))$$

$$\beta_{1,t+1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i^* - \bar{y}^*)}{\sum_{i=1}^{n}(x_i - \bar{x}^2)}$$

$$\beta_{0,t+1} = \bar{y}^* - \beta_{1,t+1}\bar{x}$$

3. continue until convergence

3.2 b

When initialization, we can just use the uncensored data and make a linear regression(using function lm()) for Y_{obs} on their corresponding x_{obs}'s. After getting the initial β_0 and β_1, we can set $\sigma_0^2 = \frac{1}{n}\sum_{i=1}^{c}(y_{obs,i} - \beta_0 - \beta_1 x_{obs,i})^2$.

3.3 c

Collaborator: Shan Gao, Yuwen Chen

```
## problem 3 simulated data

set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

x <- runif(n)
yComplete <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))

## parameters chose such that signal in data is moderately strong
## estimate divided by std error is ~ 3
mod <- lm(yComplete ~ x)
summary(mod)$coef

##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 0.5607442  0.5041346 1.112290 0.268734381
## x           2.7650812  0.8657927 3.193699 0.001889262

em = function(x, y, rate){
  #create complete data and censored data
  tau <- sort(y, decreasing = TRUE)[rate * length(y)]
  x_not_censored <- x[which(y < tau)]
  y_not_censored <- y[which(y < tau)]
  x_censored <- x[which(y > tau)]
  y_censored <- y[which(y > tau)]
  m= length(x_not_censored)
  #calculate first generation of parameter
  modd <- lm(y_not_censored ~ x_not_censored)
  beta_0 <- summary(modd)$coef[1,1]
  beta_1 <- summary(modd)$coef[2,1]
  sigma_2 <- sum((y_not_censored - beta_0 - beta_1 * x_not_censored)^2)/m
  mu_censored= c()
  iter=0
  beta1_old = 0
  # calculate next generation
  while ( abs(beta_1 - beta1_old)>0.0000005&& iter < 5000){
    beta1_old <- beta_1
    mu_censored= beta_0 + beta_1 * x_censored
    tau_star= (tau - mu_censored)/sqrt(sigma_2)
    rho = dnorm(tau_star)/(1-pnorm(tau_star))
    e_mit = mu_censored + sqrt(sigma_2)* rho
    var_mit = sigma_2 * (1+ tau_star*rho - rho^2)

    xtemp= c(x_censored, x_not_censored)
    ytemp= c(e_mit, y_not_censored)

    #get new parameters
    modd= lm(ytemp ~ xtemp)
    beta_1= summary(modd)$coef[2,1]
    beta_0= summary(modd)$coef[1,1]
    sigma_2= (sum((summary(modd)$residuals)^2) + sum(var_mit))/n
    iter = iter+1
  }
  return(c(beta_0, beta_1, sigma_2, iter))

}

#beta0 and beta1 of fitted model within full data
summary(mod)$coef

##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 0.5607442  0.5041346 1.112290 0.268734381
## x           2.7650812  0.8657927 3.193699 0.001889262

#sigma_2 of fitted model within full data
var(summary(mod)$residuals)

## [1] 5.25989

#(beta0, beta1, sigma_2, iter) result of EM algorithm within 20% data unknown
```

```r
em(x, yComplete, 0.2)

## [1]  0.5098993  2.6739004  4.5421127 13.0000000

#(beta0, beta1, sigma_2, iter) result of EM algorithm within 80% data unknown
em(x, yComplete, 0.8)

## [1]   0.3096457   2.7984693   3.6718943 144.0000000
```

From above result, I can see that my EM algorithm works very good when 20 percent data unknown. When unknown data is 80 percent, the performance is a little bit worse but still satisfactory.

3.4 d

Collaborator: Jinhui Xu, Shan Gao

```r
set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6
x <- runif(n)
yComplete <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))

# write the function to compute loglikelihood of censored
# and uncensored data
loglike <- function(para){
  beta0 = para[1]
  beta1=para[2]
  sigma2= para[3]
  rate=0.2
  tau <- sort(yComplete, decreasing = TRUE)[rate * length(yComplete)]
  # split data in to censored and uncensored
  x_uncensored <- x[which(yComplete < tau)]
  y_uncensored <- yComplete[which(yComplete < tau)]
  x_censored <- x[which(yComplete >= tau)]
  y_censored <- yComplete[which(yComplete >= tau)]

  #calculate the log likelihood of censored data and uncensored data
  lik_censored<- sum(log(1-pnorm((tau-beta0-beta1*x_censored)/sqrt(sigma2))))
  lik_uncensored <- -n*(1-rate)/2*log(2*pi*sigma2)-1/
    (2*sigma2)*crossprod(y_uncensored-beta0-beta1*x_uncensored)

  # optimize -(sum of likelihoods) to avoid negative values of log(sigma^2)
  # when using optim()
  loglikelihood <- -(lik_censored + lik_uncensored)
  return(loglikelihood)
}
# compare optim() with my own EM-algorithm
optim(c(2,2,2), loglike, method = "BFGS")

## $par
## [1] 0.4580178 2.8326082 4.6548289
##
## $value
## [1] 195.3699
##
## $counts
## function gradient
##       32       13
##
## $convergence
## [1] 0
##
## $message
## NULL

em(x, yComplete, 0.2)

## [1]  0.5098993  2.6739004  4.5421127 13.0000000
```

Comparing optim() with my own EM-algorithm, I saw that my estimate of β_0 is better than that of optim(), and my estimate of β_1, σ^2 is slightly more deviated from the true value. Moreover, optim() needs larger number of counts than that of my EM-algorithm(32 v.s. 13).