Problem Set 8

Dongyu Lang
SID: 24174288

November, 30, 2017

Question 1(a)

We first need to compute the first order derivative of the two distributions.
For exponential distribution, the first order derivative is $f^{'}(x) = -\lambda^2 e^{-\lambda x}$
For Pareto distribution, the first order derivative is $g^{'}(x) = -\frac{(\beta+1)\beta\alpha^{\beta}}{x^{\beta+2}}$
$\lim_{x\to\infty} \frac{g^{'}(x)}{f^{'}(x)} = \frac{-\frac{(\beta+1)\beta\alpha^{\beta}}{x^{\beta+2}}}{-\lambda^2 e^{-\lambda x}} = \frac{(\beta+1)\beta\alpha^{\beta}}{\lambda^2 e^{-\lambda x} x^{\beta+2}}$
And by L'Hospital's Rule, the limit goes to infinity; thus, we know that Pareto distribution decays slower than exponential distribution.

Question 1(b)

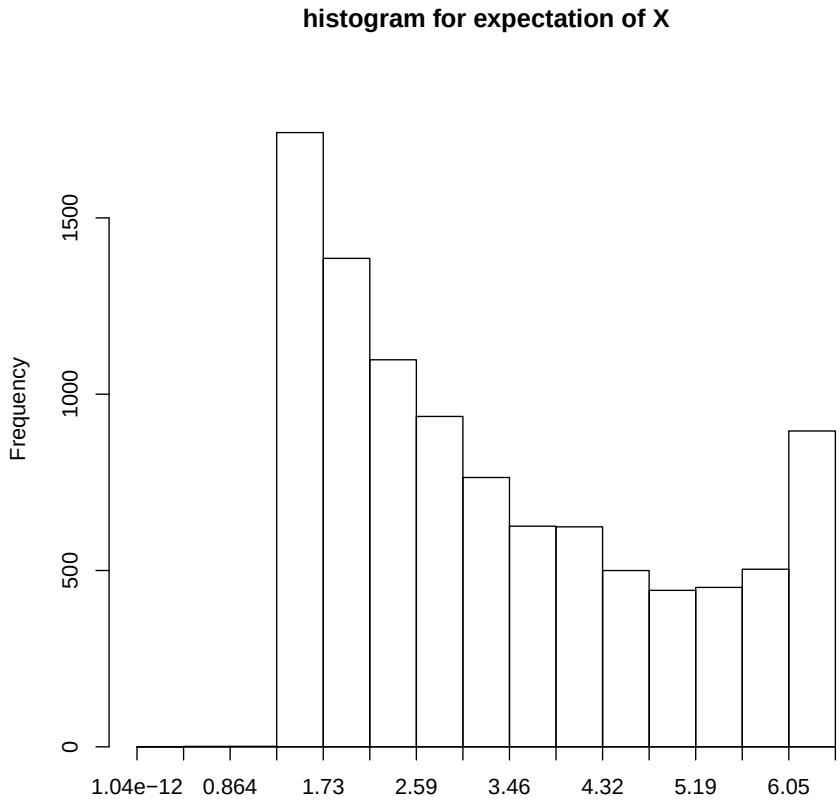
We first generate 10000 points from the proposal density, that is the Pareto distribution. Next, we create functions of f, the shifted exponential, and g, the Pareto. Then, we calculate the mean of the importance sampling.

```
## Sample from proposal density
x = rpareto(10000,location = 2, shape = 3)
## Create the fucntion f
f = function(x) {
  listf = c()
  for (i in 1:length(x)) {
    if (x[i]<2) {listf[i] = 0} else{
    listf[i] = dexp(x[i]-2, rate = 1)
    }
  }
  return (listf)
}
## Create the function g
g = function(x) {
  dpareto(x,location = 2, shape = 3)
}
```

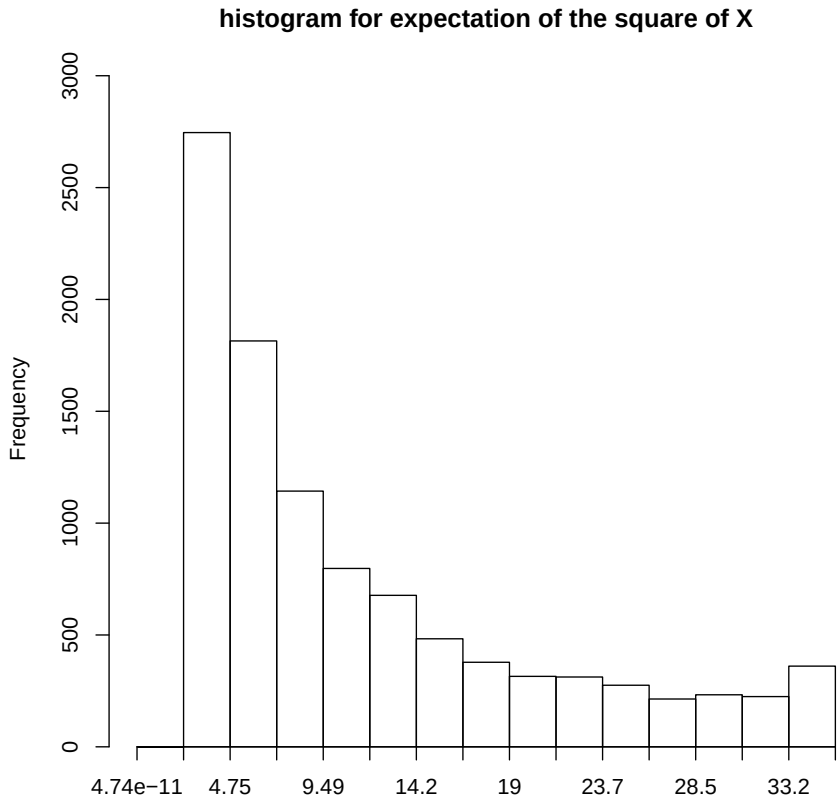
```r
## importance weight
w = f(x)/g(x)

## Expectation of X
Ex = mean(x*w)
Ex

## [1] 2.998303

## Expectation of X^2
Ex2 = mean(x^2*w)
Ex2

## [1] 10.0396
```

Thus, $E(X) = 2.9983031$ and $E(X^2) = 10.0396022$.

```r
## weighted histogram for E(X)
weighted.hist(x*w,w, main = "histogram for expectation of X")
```




```
## weighed histigram for E(X^2)
weighted.hist(x^2*w,w, main = "histogram for expectation of the square of X")
```



We can notice that the variances for both graphs are not large, but there are some extreme weights on the tail parts.

Question 1(c)

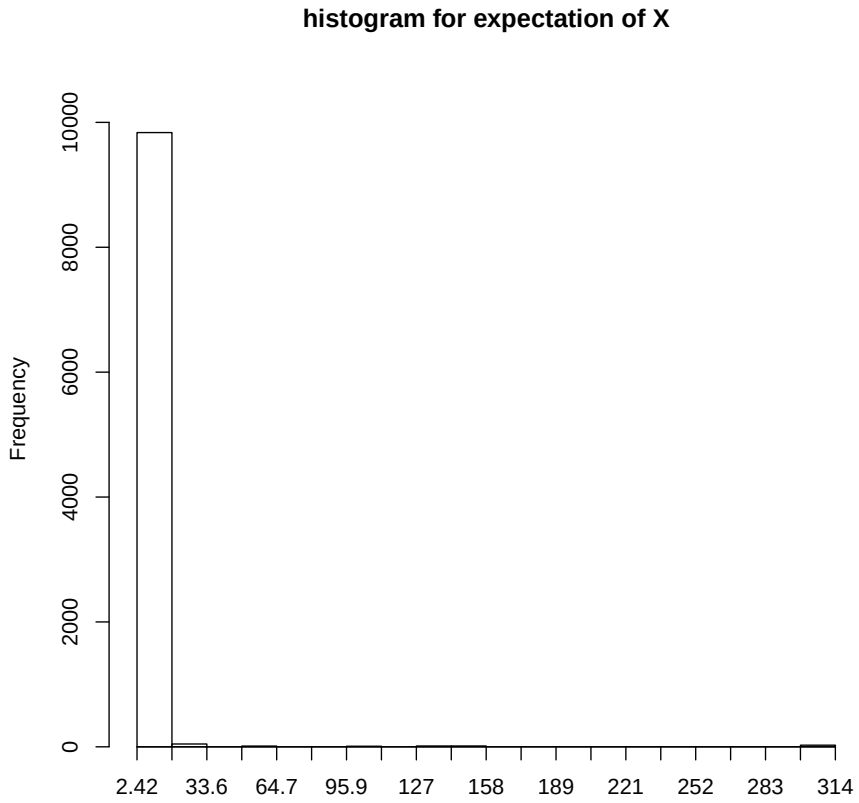
We first generate 10000 points from the proposal density, that is the shifted exponential. Next, we create functions of g, the shifted exponential, and f, the Pareto. Then, we calculate the mean of the importance sampling.

```
## Sample from proposal density
x_3 = rexp(10000)+2
## Create the fucntion f
f_3 = function(x) {
  dpareto(x,location = 2, shape = 3)
}
## Create the function g
```

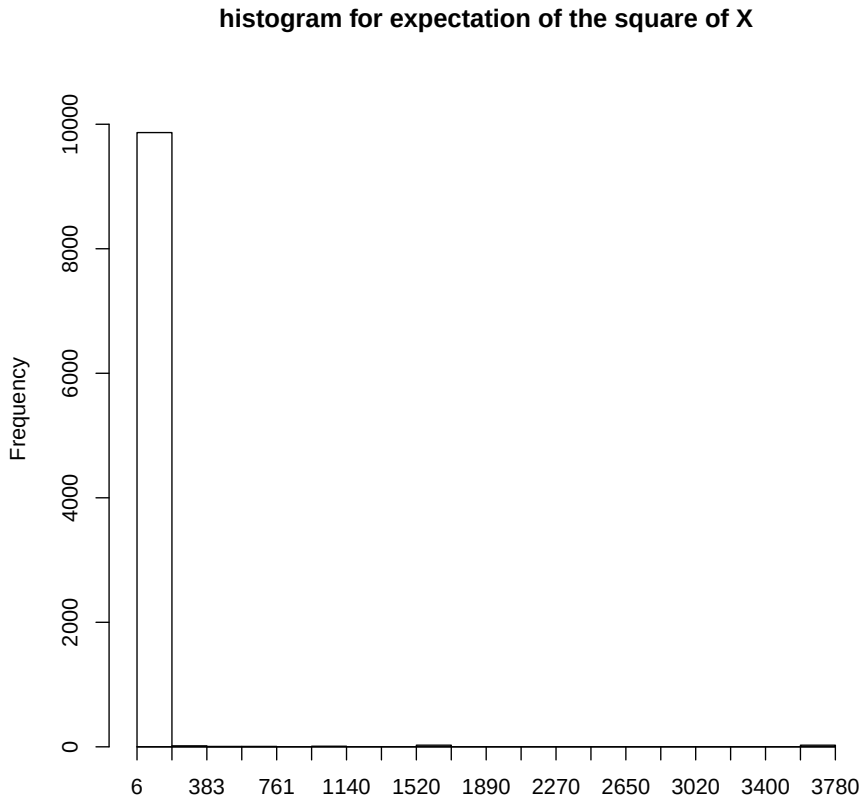
```r
g_3 = function(x) {
  listf = c()
  for (i in 1:length(x)) {
    if (x[i]<2) {listf[i] = 0} else{
    listf[i] = dexp(x[i]-2, rate = 1)
    }
  }
  return (listf)
}
## importance weight
w_3 = f_3(x_3)/g_3(x_3)

## Expectation of X
Ex_3 = mean(x_3*w_3)
Ex_3

## [1] 2.94214

## Expectation of X^2
Ex2_3 = mean(x_3^2*w_3)
Ex2_3

## [1] 10.32018
```

Thus, $E(X) = 2.9421398$ and $E(X^2) = 10.3201751$.

```r
## weighted histogram for E(X)
weighted.hist(x_3*w_3,w_3, main = "histogram for expectation of X",breaks = 20)
```




```
## weighed histigram for E(X^2)
weighted.hist(x_3^2*w_3,w_3, main = "histogram for expectation of the square of X",
              breaks = 20)
```

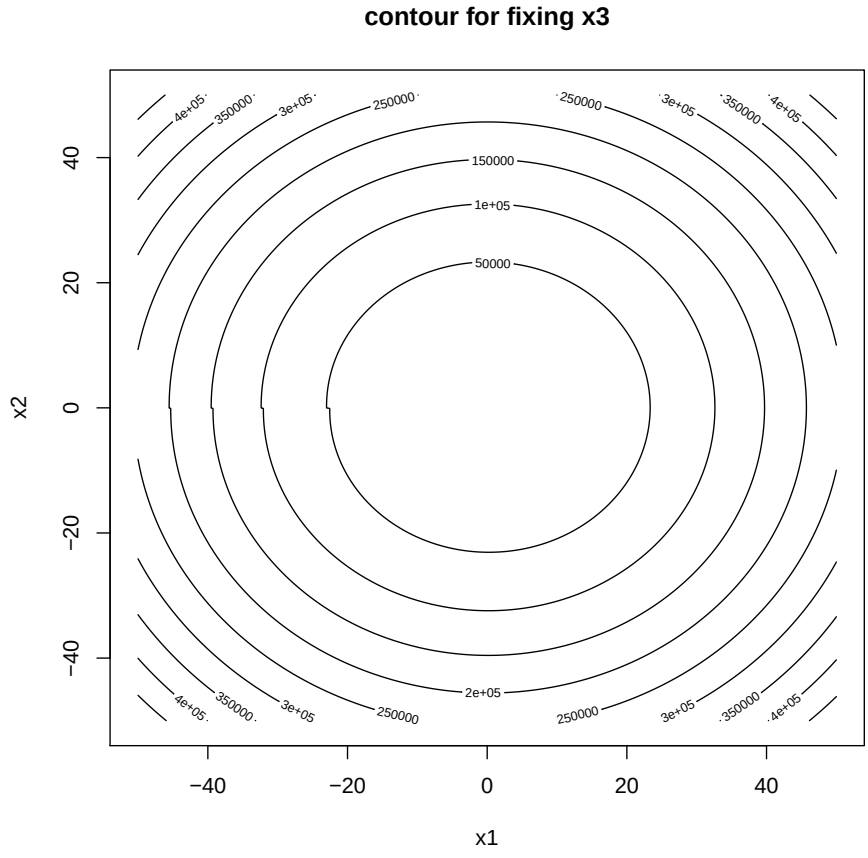


The graphs shows that the variance for each plot is large, and there are some extreme weights on the head parts.

Question 2

```r
## Copy the function from the r file
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)
}
```

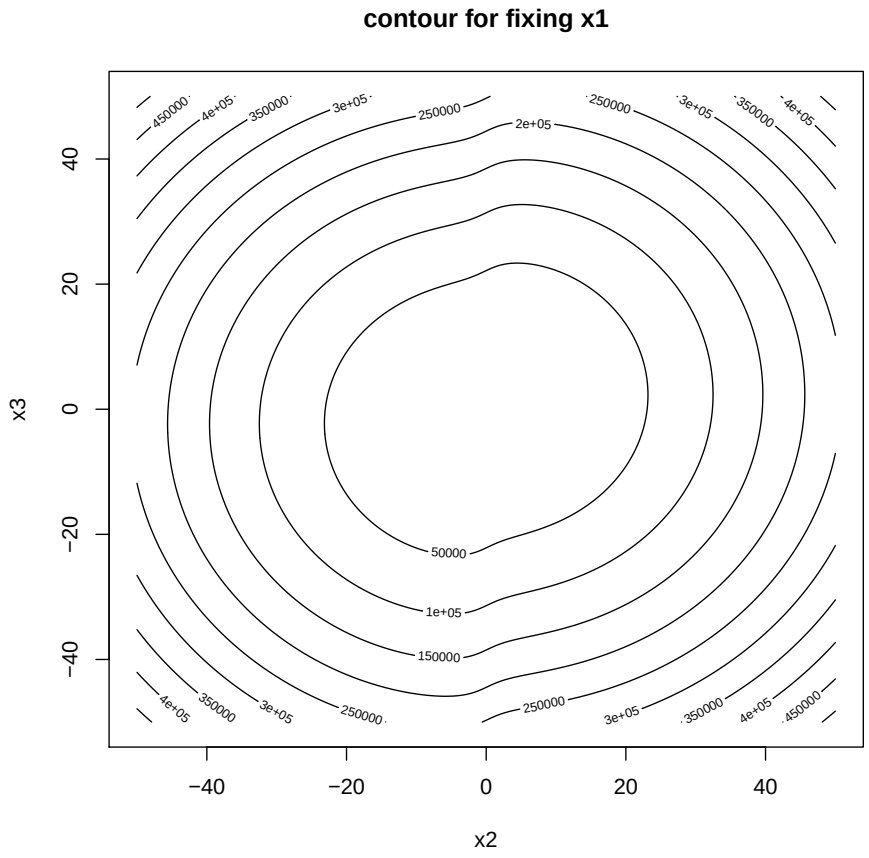
```r
## We first fix the third element to be 1
fix_3 = function(x1,x2) {
  f1 <- 10*(1 - 10*theta(x1,x2))
  f2 <- 10*(sqrt(x1^2 + x2^2) - 1)
  f3 <- 1
  return(f1^2 + f2^2 + f3^2)
}

## Create a contour plot
x1 = seq(-50,50,0.1)
x2 = seq(-50,50,0.1)
result = outer(x1,x2,fix_3)
contour(x=x1,y=x2,z=result,xlab="x1",ylab="x2",
        main="contour for fixing x3")
```



It shows that the value increases as both of the absolute values of the inputs increase.

Now Let us fix another input.

```
fix_1 <- function(x2,x3) {
  f1 <- 10*(x3 - 10*theta(3,x2))
  f2 <- 10*(sqrt(3^2 + x2^2) - 1)
  f3 <- x3
  return(f1^2 + f2^2 + f3^2)
}
x2 = seq(-50,50,0.1)
x3 = seq(-50,50,0.1)
result_fix_1 = outer(x2,x3,fix_1)
contour(x=x2,y=x3,z=result_fix_1,xlab="x2",ylab="x3",
        main="contour for fixing x1")
```



It also shows that the value increases as both of the absolute values of inputs increase.
Now let us do the optimization

```r
## create different starting points
x1 = runif(100,-20,20)
x2 = runif(100,-50,50)
x3 = runif(100,-30,30)

## Get the estimated minimum point for each starting point
localest = c()
for(i in 1:100){
  localest[[i]] = nlm(f,p=c(x1[i],x2[i],x3[i]))$estimate
}

## Get the minimum value for each starting point
localmin = c()
for (i in 1:100){
  localmin[[i]] = nlm(f,p=c(x1[i],x2[i],x3[i]))$minimum
}

## Create a dataframe containing the estimate points and minimum values
locals = as.data.frame(matrix(unlist(localest),
                              nrow=100,byrow = T))
locals = cbind(locals,localmin)
colnames(locals) = c("x1", "x2","x3","min value")
head(locals)
```

```
##          x1            x2            x3    min value
## 1 1.0000000  9.101205e-12  5.210359e-11 4.975789e-19
## 2 0.9999995 -8.222869e-05 -1.300754e-04 1.700793e-08
## 3 1.0000000  7.285384e-10  1.694370e-09 4.330947e-17
## 4 1.0000000  6.209721e-10  5.825112e-10 1.697586e-17
## 5 0.9999995 -8.223219e-05 -1.300806e-04 1.700930e-08
## 6 1.0000000  3.233867e-10  1.493339e-09 1.059401e-16
```

```r
## unique values of local minimum
length(unique(locals$`min value`))
```

```
## [1] 100
```

```r
## the minimum point in these 100 values
amongmin = locals[which.min(locals$`min value`),1:3]
amongmin
```

```
##    x1           x2           x3
## 93  1 7.751158e-11 1.114233e-10
```

We tried 100 different starting points, and they all returned different estimated minimum points and values. The minimum point among these 100 values is

$1.00000000000541, 7.75115840552365e-11, 1.11423338360279e-10$

Question 3(a)

Let's first define a $\delta_i = 1$, if Y_i is not censored, and $\delta_i = 0$, if Y_i is censored.

Thus,
$Q(\theta|\theta_t) = E(logL(\theta|Y)|x, \theta_t)$
$= -nlog\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n} E[(Yi - \beta_0 - \beta_1 x_i)^2|x, \theta_t]$

For data that is not censored, the conditional expectation is just $(Y_i - \beta_0 - \beta_1 x_i)^2$. For data that is censored, the conditional expectation is $E(Y_i^2|Y_i > \tau) - 2(\beta_0 + \beta_1 x_i)E(Y_i|Y_i > \tau) + (\beta_0 + \beta_1 x_i)^2$

Thus, $Q(\theta|\theta_t)$
$= -nlog\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n} \delta_i(y_i - (\beta_0 + \beta_1 x_i))^2 -$
$\frac{1}{2\sigma^2}\sum_{i=1}^{n}(1-\delta_i)[E_{\theta_t}(y_i^2|y_i > \tau) - 2(\beta_0 + \beta_1 x_i)E_{\theta_t}(y_i|y_i > \tau) + (\beta_0 + \beta_1 x_i)^2]$

In addition, we know that,
$E_{\theta_t}(y_i|y_i > \tau) = \beta_{0t} + \beta_{1t}x_i + \sigma_t\rho(\tau^\star)$
$E_{\theta_t}(y_i^2|y_i > \tau) = \sigma_t^2(1 + \tau^\star\rho(\tau^\star) - \rho(\tau^\star)^2) + (\beta_{0t} + \beta_{1t}x_i + \sigma_t\rho(\tau^\star))^2$
$\rho(\tau^\star) = \frac{\phi(\tau^\star)}{1-\Phi(\tau^\star)}$
$\tau^\star = (\tau - (\beta_{0t} + \beta_{1t}x_i))/\sigma_t$

Now, let us define two new variables:
$\tilde{y}_i = \delta_i y_i + (1-\delta_i)(\beta_{0t} + \beta_{1t}x_i + \sigma_t\rho(\tau^\star))$
$\tilde{r}_i = \delta_i(y_i - (\beta_{0t} + \beta_{1t}x_i))^2 + \sigma_t^2(1-\delta_i)(1 + \tau^\star\rho(\tau^\star))$

Thus, maximizing $Q(\theta|\theta_t)$ is equivalent to minimizing $\sum_{i=1}^{n}(\tilde{y}_i - \beta_0 - \beta_1 x_i)^2$.

The EM algorithm is:
E step: calculate
$\tilde{y}_i = \delta_i y_i + (1-\delta_i)(\beta_{0t} + \beta_{1t}x_i + \sigma_t\rho(\tau^\star))$
$\tilde{r}_i = \delta_i(y_i - (\beta_{0t} + \beta_{1t}x_i))^2 + \sigma_t^2(1-\delta_i)(1 + \tau^\star\rho(\tau^\star))$

M step: Get θ_{t+1} from minimizing $\sum_{i=1}^{n}(\tilde{y}_i - \beta_0 - \beta_1 x_i)^2$.
Get σ_{t+1}^2 from $\sigma_{t+1}^2 = \frac{1}{n}\sum_{i=1}^{n}\tilde{r}_i$

Return to E step.

Question 3(b)

The starting value of y_i is only the non-censored y_i, and we omit the censored y. We then do a OLS.
Thus, $\beta_{1,0} = \frac{Cov(x,y)}{Var(x)}$
$\beta_{0,0} = \bar{y} - \beta_{1,0}\bar{x}$
$\sigma_0^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_{1,0}x_i - \beta_{0,0})^2$

Question 3(c)

```
## Generate the data
set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

x <- runif(n)
yComplete <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))
## create modest and high proportion of censored data
y_modest = yComplete
y_high = yComplete
for (i in 1:length(yComplete)) {
  if (yComplete[i]>as.numeric(quantile(yComplete,0.8))){
    y_modest[i] = as.numeric(quantile(yComplete,0.8))
  }
}

for (i in 1:length(yComplete)) {
  if (yComplete[i]>as.numeric(quantile(yComplete,0.2))){
    y_high[i] = as.numeric(quantile(yComplete,0.2))
  }
}

## Perform EM algorithm for modest proportion of censored data

## Get initial value for theta
y_init = y_modest[y_modest!=max(y_modest)]
x_init = x[y_modest!=max(y_modest)]
beta0 = as.numeric(lm(y_init~x_init)$coeff[1])
beta1 = as.numeric(lm(y_init~x_init)$coeff[2])
sigma0 = sqrt(mean((y_init - beta1*x_init - beta0)^2))
rss0 =  sum(resid(lm(y_init~x_init))^2)
beta_mod0 = c(beta0)
beta_mod1 = c(beta1)
sigma_mod = c(sigma0)
rss_mod = c(rss0)

## Performing EM algorithm
noncensored_index = (1:length(x))[y_modest!= max(y_modest)]
index = rep(0,length(x))
index[noncensored_index] = 1
iter = 0
```

```r
##Suppose the maximum iteration is 2000, and the error to stop iteration is 1e-8
for (i in 1:2000){
  ## E-step
  tau_star = (max(y_modest)-(beta_mod0[i]+beta_mod1[i]*x))/sigma_mod[i]
  rho_tau = dnorm(tau_star)/(1-pnorm(tau_star))
  y_tilde = index*y_modest + (1-index)*(beta_mod0[i]+beta_mod1[i]*x+sigma_mod[i]*rho_tau)
  r_tilde = index*(y_modest-(beta_mod0[i]+beta_mod1[i]*x))^2 +
    sigma_mod[i]^2*(1-index)*(1+ tau_star*rho_tau)
  ## M-step
  lm_M = lm(y_tilde~x)
  sigma_updated = sqrt(mean(r_tilde))
  ## Update theta
  beta_mod0[i+1] = as.numeric(lm_M$coeff[1])
  beta_mod1[i+1] = as.numeric(lm_M$coeff[2])
  sigma_mod[i+1] = sigma_updated
  rss_mod[i+1] = sum(resid(lm_M)^2)
  iter = iter + 1
  if (abs(rss_mod[i+1]-rss_mod[i]) <= 1e-8){
    break
  }
}

## number of iteration
iter-1

## [1] 20

## the estimated theta
##beta 0
beta_mod0[iter]

## [1] 0.4566128

##beta 1
beta_mod1[iter]

## [1] 2.824108

##variance
sigma_mod[iter]^2

## [1] 4.618876
```

We can notice that it is a bit different from the true θ; however, the estimated is close to the true one.

Now, we consider the high proportion of the censored data

```r
## Perform EM algorithm for high proportion of censored data

## Get initial value for theta
y_init_high = y_high[y_high!=max(y_high)]
x_init_high = x[y_high!=max(y_high)]
beta0 = as.numeric(lm(y_init_high~x_init_high)$coeff[1])
beta1 = as.numeric(lm(y_init_high~x_init_high)$coeff[2])
sigma0 = sqrt(mean((y_init_high - beta1*x_init_high - beta0)^2))
rss0 =  sum(resid(lm(y_init_high~x_init_high))^2)
beta_mod0 = c(beta0)
beta_mod1 = c(beta1)
sigma_mod = c(sigma0)
rss_mod = c(rss0)

## Performing EM algorithm
noncensored_index = (1:length(x))[y_high!= max(y_high)]
index = rep(0,length(x))
index[noncensored_index] = 1
iter = 0
##Suppose the maximum iteration is 2000, and the error to stop iteration is 1e-8
for (i in 1:2000){
  ## E-step
  tau_star = (max(y_high)-(beta_mod0[i]+beta_mod1[i]*x))/sigma_mod[i]
  rho_tau = dnorm(tau_star)/(1-pnorm(tau_star))
  y_tilde = index*y_high + (1-index)*(beta_mod0[i]+beta_mod1[i]*x+sigma_mod[i]*rho_tau)
  r_tilde = index*(y_high-(beta_mod0[i]+beta_mod1[i]*x))^2 +
    sigma_mod[i]^2*(1-index)*(1+ tau_star*rho_tau)
  ## M-step
  lm_M = lm(y_tilde~x)
  sigma_updated = sqrt(mean(r_tilde))
  ## Update theta
  beta_mod0[i+1] = as.numeric(lm_M$coeff[1])
  beta_mod1[i+1] = as.numeric(lm_M$coeff[2])
  sigma_mod[i+1] = sigma_updated
  rss_mod[i+1] = sum(resid(lm_M)^2)
  iter = iter + 1
  if (abs(rss_mod[i+1]-rss_mod[i]) <= 1e-8){
    break
  }
}

## number of iteration
iter-1


## [1] 250
```

```r
## the estimated theta
##beta 0
beta_mod0[iter]

## [1] 0.3126394

##beta 1
beta_mod1[iter]

## [1] 2.87922

##variance
sigma_mod[iter]^2

## [1] 3.841964
```

In this situation, it takes more iterations; however, the estimated θ is still close to the true one.

Question 3(d)

```r
## Let us first write the log-likelihood function for modest proportion
lntheta_mod = function(input){
  y = y_modest
  noncensored_index = (1:length(x))[y!= max(y)]
  index = rep(0,length(x))
  index[noncensored_index] = 1
  beta0 = input[1]
  beta1 = input[2]
  sigma = input[3]
  tau_star = (max(y)-(beta0+beta1*x))/sigma
  rho_tau = dnorm(tau_star)/(1-pnorm(tau_star))
  ## Create the log-likehood
  loglike = -length(y)*log(sigma)-
    1/(2*sigma^2)*sum(index*(y-(beta0+beta1*x))^2)-
    1/(2*sigma^2)*sum((1-index)*((sigma^2*(1+tau_star*rho_tau-rho_tau^2)+
                          (beta0+beta1*x+sigma*rho_tau)^2)-
                      2*(beta0+beta1*x)*(beta0+beta1*x+sigma*rho_tau)+
                          (beta0+beta1*x)^2))
  return(-loglike)
}
## Perform an optimization
opt_mod = optim(c(1,2,sqrt(6)),lntheta_mod, method = "BFGS",control = list(fnscale = 1))

## Warning in log(sigma):  NaNs produced
```

```
## Warning in log(sigma):  NaNs produced

## the estimated theta
opt_mod$par

## [1] 0.4405234 2.4942613 1.8205455

## number of iteration
as.numeric(opt_mod$counts[2])

## [1] 8
```

The result shows that it takes less iterations than part(c), and the result is also close to the true θ.

Now let us consider the high proportion of censored data.

```
## Let us first write the log-likelihood function for modest proportion
lntheta_high = function(input){
  y = y_high
  noncensored_index = (1:length(x))[y!= max(y)]
  index = rep(0,length(x))
  index[noncensored_index] = 1
  beta0 = input[1]
  beta1 = input[2]
  sigma = input[3]
  tau_star = (max(y)-(beta0+beta1*x))/sigma
  rho_tau = dnorm(tau_star)/(1-pnorm(tau_star))
  ## Create the log-likehood
  loglike = -length(y)*log(sigma)-
    1/(2*sigma^2)*sum(index*(y-(beta0+beta1*x))^2)-
    1/(2*sigma^2)*sum((1-index)*((sigma^2*(1+tau_star*rho_tau-rho_tau^2)+
                        (beta0+beta1*x+sigma*rho_tau)^2)-
                    2*(beta0+beta1*x)*(beta0+beta1*x+sigma*rho_tau)+
                        (beta0+beta1*x)^2))
  return(-loglike)
}
## Perform an optimization
opt_mod = optim(c(1,2,sqrt(6)),lntheta_high, method = "BFGS",control = list(fnscale = 1))

## Warning in log(sigma):  NaNs produced
## Warning in log(sigma):  NaNs produced
## Warning in log(sigma):  NaNs produced
## Warning in log(sigma):  NaNs produced
## Warning in log(sigma):  NaNs produced
## Warning in log(sigma):  NaNs produced
```

```r
## the estimated theta
opt_mod$par
```

```
## [1] -0.4664671  0.9392315  0.6512491
```

```r
## number of iteration
as.numeric(opt_mod$counts[2])
```

```
## [1] 13
```

It is not accurate in this situation, even though the iteration is much smaller.

Collaboration

In this problem set, I worked with Gaowei Chen.