STAT 243: Promlem Set 8

Xinyue Zhou

December 7, 2015

1 Question 1

1.1 a

I used following three steps to generate the dataset and evaluate standard linear model against robust linear model.

- First of all, a testing data set should be generate here. To get the general results, the number of variables, saying X_i should varies. Our data set is generate from the following model:

$$Y_i = \beta_0 + \Sigma_{k=1}^{p} \beta_k X_i + \epsilon_i \tag{1}$$

$$\epsilon_i \sim \begin{cases} Normal & \text{w.p. } \alpha \\ distribution_2 & \text{w.p. } 1-\alpha \end{cases} \tag{2}$$

  Show in the above model, α percentage of data is from regular, or expected, data set, and $1-\alpha$ are outliers, where α smaller than 50% . β 's are known at beginning, so after ϵ_i 's are decided and a sequence of X_i's, Y_i's are got from summation as shown in (1). Notice that $distribution_2$ should be very different from Normal distribution in the first case.

- Fit the model using two different method.

- Evaluate the performance of two models using two measurements: 1) absolute predict error(MAPE); 2) coverage of prediction intervals (CPI)

$$MAPE = \frac{1}{N} \left| Y_i - \hat{Y}_i \right| \tag{3}$$

$$CPI = \frac{1}{N} \mathbb{1}\{Y_i \in C_i\} \tag{4}$$

  In the above equations, CI are prediction intervals calculated using bootstrap. The input of bootstrap is the new data set and the output is bunch of $\hat{Y}$ values calculating from bootstrap samples. The smaller MAPE and the larger CPI is, the better performance of the model will be.

1.2 b

Here, I create a concrete case to test the performance of standard linear model against robust linear model (using R package *MASS*, and function *rlm*).

```
library(MASS)
#generate data set
dataset <- function(p=0,beta,alpha=0,N){
  X<- matrix(numeric(N*(p+1)),ncol = p+1)
  X[,1] = rep(1,N)
  for (i in 2:ncol(X)) {X[,i] = runif(N,-10,10)}
  tmp <-(runif(N)<alpha)
  e <- numeric(N)
```

```
  #draw outlier
  e[tmp]<- rnorm(sum(tmp),0,100)
  #draw regular data
  e[tmp==FALSE] <- rnorm(N-sum(tmp),0,1)
  Y = X%*%beta+e
  return (cbind(Y,X[,-1]))
}


data = dataset(3,c(1,2,3,4),0.1,100)
fit1 = lm(data[,1]~data[,-1])
fit2 = rlm(data[,1]~data[,-1])
fit1$coefficients

## (Intercept) data[, -1]1 data[, -1]2 data[, -1]3
##    3.032228    1.824681    3.164408    3.354847

fit2$coefficients

## (Intercept) data[, -1]1 data[, -1]2 data[, -1]3
##    1.127478    1.976895    2.990164    4.015110

#create a new obs set, and store new Y
new_data = dataset(3,c(1,2,3,4),0.1,20)
new_Y = new_data[,1]
new_X = cbind(rep(1,20),new_data[,-1])

#calculate the fitted values
Y_fit1 = new_X%*%fit1$coefficients
Y_fit2 = new_X%*%fit2$coefficients

#evaluate the predict absolute error here.
Mape1 = mean(abs(Y_fit1-new_Y))
Mape2 = mean(abs(Y_fit2-new_Y))
c(Mape1 = Mape1,Mape2 = Mape2)

##    Mape1    Mape2
## 17.98179 15.58244
```

As *Mape2* is smaller than *Mape1*, the robust regression is much better for fitting data set with bunch of outliers.

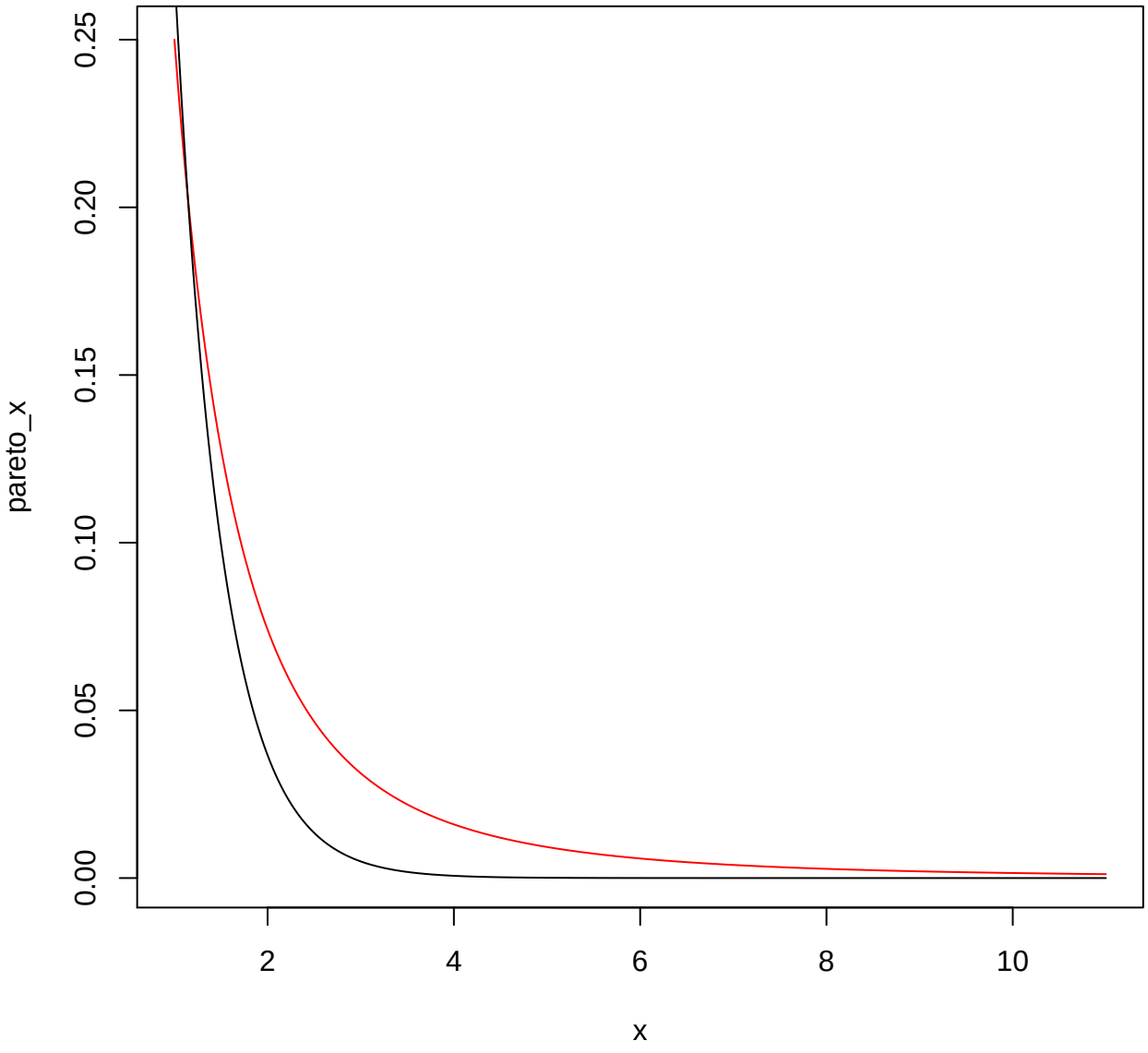
2 Question 2

2.1 a

To visualize the tail shape of exponential distribution and pareto distribution, I plot these two distributions together. Note, this too density is using the same rate (rate=1).

```
require(actuar)

## Loading required package:  actuar
##
## Attaching package:  'actuar'
##
## The following object is masked from 'package:grDevices':
##
##    cm
```

```r
alpha = 1
beta = 2
x = seq(alpha,alpha+10,length = 500)
pareto_x = dpareto(x,scale = alpha,shape = beta)
plot (x, pareto_x,type="l",col = "red")
lines(x,dexp(x,rate=beta))
```



2.2 b

As f is the exp(1) shift 2 units to right, then the density of f is:

$$f(x) = e^{-(x-2)} \tag{5}$$

For the sampling density, $g \sim Pareto(\alpha = 2, \beta = 3)$, then:

$$g(x) = \frac{\beta\alpha^{\beta}}{x^{\beta+1}} = \frac{24}{x^4} \tag{6}$$

To approximate the expectation of f distribution, importance sampling can be used here:

$$E_f(h(X)) = \int_{x\in\chi} h(x)\frac{f(x)}{g(x)}g(x)dx \tag{7}$$

$$E_f(X) \approx \frac{1}{m}\Sigma_{i=1}^{m}X_i \times \frac{f(X_i)}{g(X_i)} \tag{8}$$

$$E_f(X^2) \approx \frac{1}{m}\Sigma_{i=1}^{m}X_i^2 \times \frac{f(X_i)}{g(X_i)} \tag{9}$$

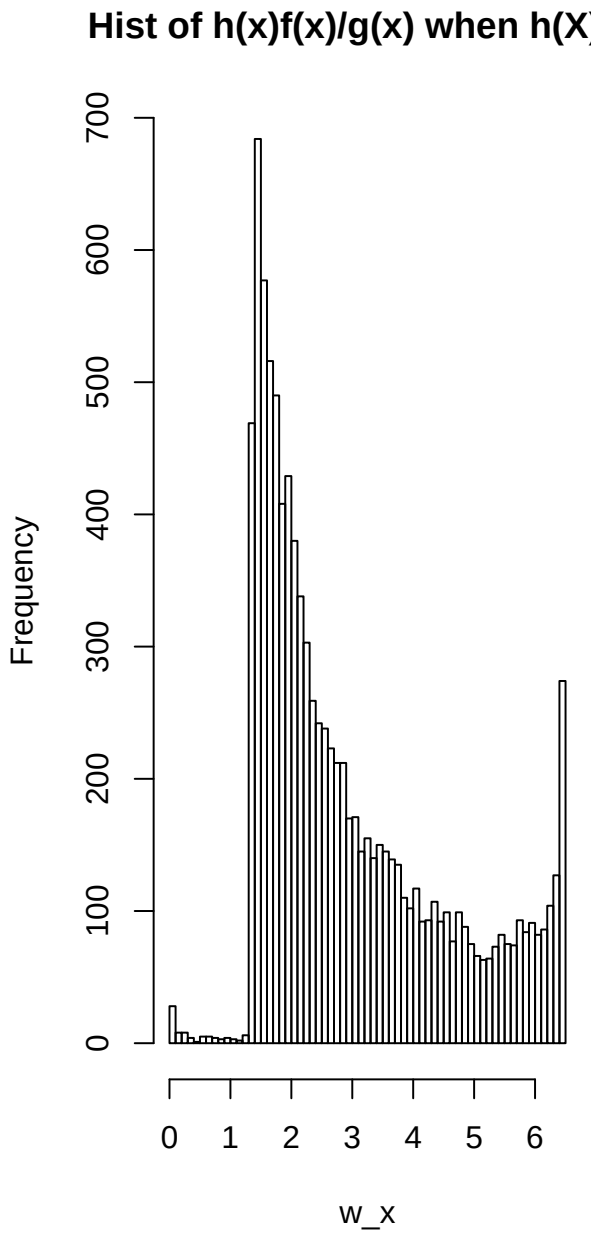
Algorithm:

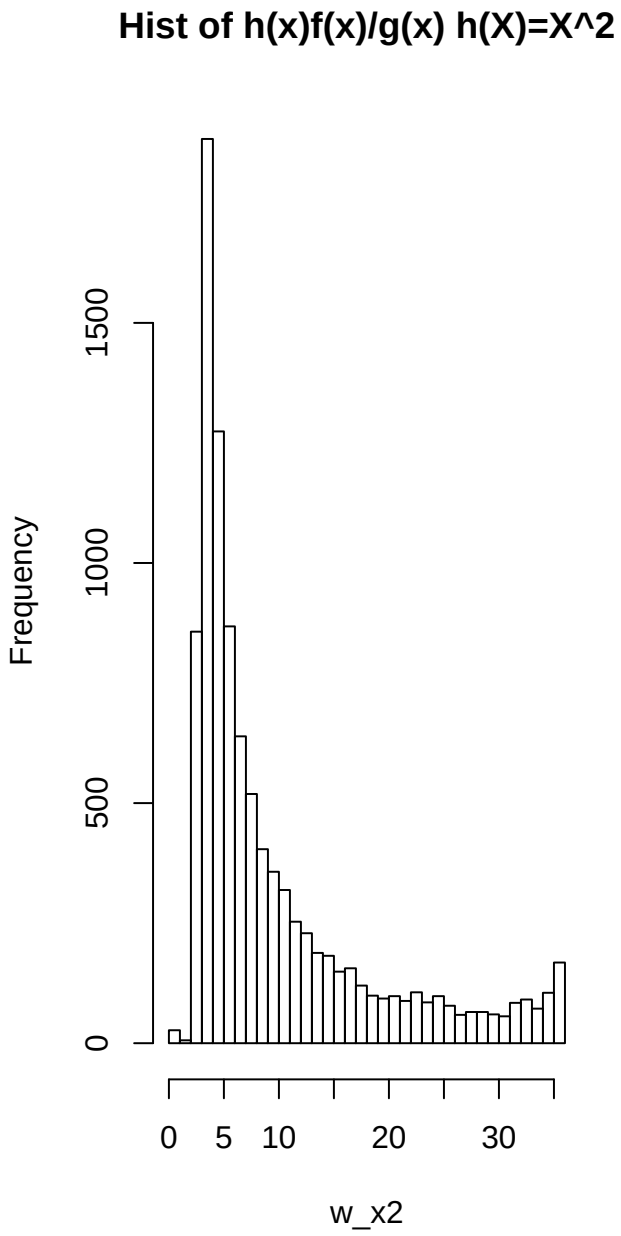
- First draw m=10000 samples from *g(x)*.
- Estimate $E_f(X)$ and $E_f(X^2)$ using (7), (8).

Note that the variance of $\hat{\mu}$ can be get using:

$$Var(\hat{\mu}) = \frac{1}{m}Var\left(h(X)\frac{f(X)}{g(X)}\right) \tag{10}$$

```
library(actuar)
#Note the formular for pareto is slightly different from the one in the prob
m = 10000
alpha = 2
beta = 3
#draw samples
x = rpareto(m,scale = alpha,shape = beta)+alpha
f_over_g = exp(-x+2)/(24/x^4)
#E(x)
w_x = f_over_g *x
w_x2 = f_over_g *x^2
mu_est = mean(w_x)
mu_est

## [1] 2.964115

#E(x^2)
sec_mom_est = mean(w_x2)
sec_mom_est

## [1] 9.809756

#draw the histogram of h(x)f(x)/g(x)
par(mfrow = c(1,2))
hist(w_x,breaks = 50, main = "Hist of h(x)f(x)/g(x) when h(X)=X")
hist(w_x2,breaks = 50, main = "Hist of h(x)f(x)/g(x) h(X)=X^2")
```






```
#calculate the variance of estimate:
var_mu = var(w_x)/m
var_sec_mom = var(w_x2)/m
c(var_mu = var_mu, var_second_mom = var_sec_mom)

##         var_mu var_second_mom
##   0.0002334769   0.0073292860
```

2.3 c

Repeat the same thing as in part **b**, just exchange the sampling distribution and target distribution. The results is as followed.

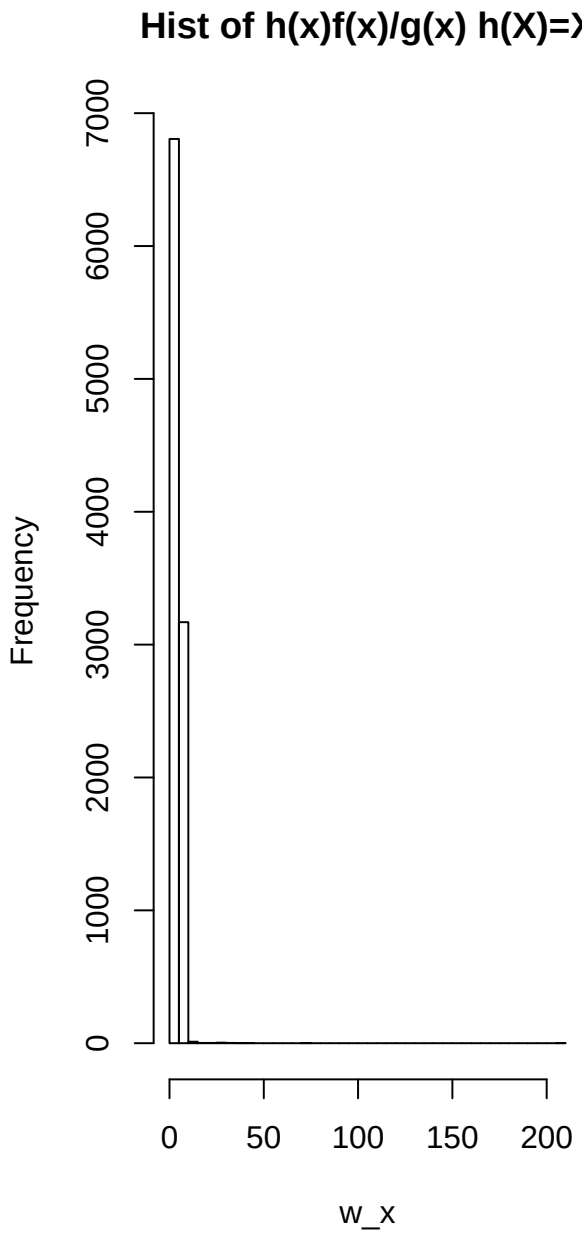
```r
#draw samples
x = rexp(m)+1
f_over_g = (24/x^4)/exp(-x+2)
#E(x)
w_x = f_over_g *x
w_x2 = f_over_g *x^2
mu_est = mean(w_x)
mu_est
```

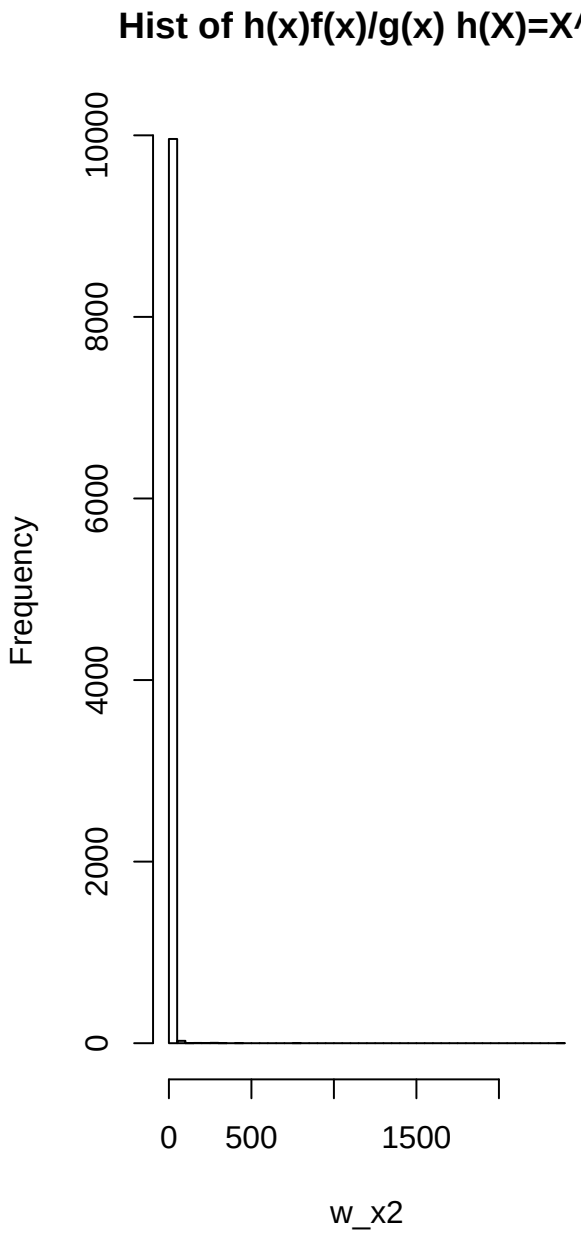
```
## [1] 4.419193
```

```r
#E(x^2)
sec_mom_est = mean(w_x2)
sec_mom_est
```

```
## [1] 8.23994
```

```r
#draw the histogram of h(x)f(x)/g(x)
par(mfrow = c(1,2))
hist(w_x,breaks = 50, main = "Hist of h(x)f(x)/g(x) h(X)=X")
hist(w_x2,breaks = 50, main = "Hist of h(x)f(x)/g(x) h(X)=X^2")
```






```r
#calculate the variance of estimate:
var_mu = var(w_x)/m
var_sec_mom = var(w_x2)/m
c(var_mu = var_mu, var_second_mom = var_sec_mom)

##         var_mu var_second_mom
##    0.000873541    0.070927466
```

Compare the variance of $\hat{\mu}$ in both Part**b** and Part**c**, the one in Part**c** is much larger than that in b. The result is consistent with the fact that selected sample distribution should own a heavier tail than the target distribution, which can promise a better result of estimation.

3 Question 3

3.1 a

Generally speaking, the EM algorithm is as followed:

- E step: Compute $Q(\theta|\theta_t)$, ideally calculating the expectation over the missing data in closed form. Note that log $L(\theta|Y)$ is a function of θ so $Q(\theta|\theta_t)$ will involve both θ and θ_t.
- M step: Maximize $Q(\theta|\theta_t)$ with respect to θ, finding θ_t.
- Continue until convergence.

To implement it on this setup, first of all, I calculate $Q(\theta|\theta_t)$.

$$\begin{aligned} L(\boldsymbol{\beta}|Y,Z) &= \prod_i f(Y_i, Z_i|\boldsymbol{\beta}) \\ &= \prod_i f_Z(Z_i|\boldsymbol{\beta}) f_Y(Y_i|Z_i,\boldsymbol{\beta}) \\ &= \prod_i \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2}(Z - X_i^T\boldsymbol{\beta})^2\right) \mathbb{1}\{Z_i > 0\}^{Y_i}(1 - \mathbb{1}\{Z_i > 0\})^{(1-Y_i)} \end{aligned} \tag{11}$$

Therefore:

$$\begin{aligned} l(\boldsymbol{\beta}|Y,Z) &= log(L(\boldsymbol{\beta}|Y,Z)) \\ &= c + \sum_i \left(-\frac{1}{2}(Z - X_i^T\boldsymbol{\beta})^2\right) + \sum log\left(\mathbb{1}\{Z_i > 0\}^{Y_i}(1 - \mathbb{1}\{Z_i > 0\})^{(1-Y_i)}\right) \end{aligned} \tag{12}$$

Notice that the last term of (16) is always 0. As we can ignore the constant term when optimize the function, I just ignore it from now on.

$$\begin{aligned} Q(\boldsymbol{\beta}_t|\boldsymbol{\beta}_t) &= E\left[l(\boldsymbol{\beta}|y,z)|\cdot\right] \\ &= \sum_i E\left(-\frac{1}{2}(Z - X_i^T\boldsymbol{\beta})^2|\cdot\right) \\ &= \frac{1}{2}E\left(Z_i^2|\cdot\right) + X_i^T\boldsymbol{\beta}E\left(Z_i|\cdot\right) - \frac{1}{2}\left(X_i^T\boldsymbol{\beta}\middle|\cdot\right)^2 \end{aligned} \tag{13}$$

Then maximize $Q(\boldsymbol{\beta}|\boldsymbol{\beta}_t)$ to get $\hat{\boldsymbol{\beta}}_{t+1}$

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{t+1} &= \mathbf{Argmax}_\beta Q(\boldsymbol{\beta}|\boldsymbol{\beta}_t) \\ &= \mathbf{Argmin}_\beta \frac{1}{2}\left(X_i^T\boldsymbol{\beta}\middle|\cdot\right)^2 - X_i^T\boldsymbol{\beta}E\left(Z_i|\cdot\right) \end{aligned} \tag{14}$$

Take derivative and solve (14), I get:

$$\hat{\boldsymbol{\beta}}_{t+1} = (X^TX)^{-1}X^TE(Z|\cdot) \tag{15}$$

See the equation (16) is very like estimating parameter of multiple linear regression under OLS.

From Johnson and Kotz bibles on distributions I found that :

$$E[Z_i|Y_i, X_i, \boldsymbol{\beta}_t] = \begin{cases} E[Z_i|Z_i > 0, X_i, \boldsymbol{\beta}_t] = X_i^T\boldsymbol{\beta}_t + \frac{\phi(-X_i^T\boldsymbol{\beta}_t)}{1-\Phi(-X_i^T\boldsymbol{\beta}_t)} & \text{when } Y_i = 1 \\ E[Z_i|Z_i < 0, X_i, \boldsymbol{\beta}_t] = X_i^T\boldsymbol{\beta}_t - \frac{\phi(-X_i^T\boldsymbol{\beta}_t)}{\Phi(-X_i^T\boldsymbol{\beta}_t)} & \text{when } Y_i = 0 \end{cases}$$

3.2 b

See the equation (22) is pretty like estimating parameter of multiple linear regression under OLS, and $E(Z|\cdot)$ is closely related with Z_i. Therefore, it is reasonable to following term to initialize the algorithm.

$$\beta_0 = (X^TX)^{-1}X^TY \tag{16}$$

3.3 c

Imprement the algorithm under this setup:

```
options(digits = 4)
#generate data set
n = 100
beta = c(1,1,0,0)
set.seed(0)
Xs = cbind(rep(1,n),matrix(rnorm(3*n),ncol = 3))
P_z = pnorm(0,mean = Xs%*%beta, lower.tail = FALSE)
Y = sapply(P_z, function(x)return(rbinom(1, 1, x)))

JK <- function(Xs, Y, beta){
  ret = numeric(length(Y))
  Xbt = Xs %*% beta
  mask = as.logical(Y)
  ret[mask] = Xbt[mask]+dnorm(-Xbt[mask])/(1-pnorm(-Xbt[mask]))
  ret[!mask] = Xbt[!mask]-dnorm(-Xbt[!mask])/pnorm(-Xbt[!mask])
  return(ret)
}

regr_beta <- function(Xs,JKvalue){
  return(lm(JKvalue~Xs[, 2:4])$coefficients)
}

EM_beta <- function(beta0,Xs, Y, rate = 1e-06, step = 1000){
  beta_t = beta0
  beta_t_1 = regr_beta(Xs, JK(Xs,Y,beta_t))
  it = 0
  while(max(abs(beta_t_1-beta_t)) > rate && it < step){
      beta_t = beta_t_1
      beta_t_1 = regr_beta(Xs,JK(Xs,Y,beta_t))
      it = it + 1
  }
  list (beta_hat = beta_t_1, steps = it)
}

EM_beta(lm(Y~Xs[, 2:4])$coefficients,Xs,Y)

## $beta_hat
## (Intercept)  Xs[, 2:4]1  Xs[, 2:4]2  Xs[, 2:4]3
##    1.586535    1.372024   -0.007274    0.636257
##
## $steps
## [1] 181
```

3.4 d

In the previous section, I found that it takes EM method 182 iteration to converge. Next BFGS method was directly applied to maximize the log-likelihood of the observed data.Derive log-likelihood as followed:

$$f_Y(Y|X,\boldsymbol{\beta}) = \sum_i \Phi(X_i^T\boldsymbol{\beta})^{Y_i}(1-\Phi(X_i^T\boldsymbol{\beta}))^{1-Y_i}$$

$$l(\boldsymbol{\beta}|Y,Z) = \sum_i Y_i log(\Phi(X_i^T\boldsymbol{\beta})) + (1-Y_i)log(1-\Phi(X_i^T\boldsymbol{\beta})) \tag{17}$$

Then, *optim()* is used to maximize log-likelihood function.

```
log_lik <- function(beta){
sum(Y*log(pnorm(Xs %*% beta)) + (1-Y)*log(1-pnorm(Xs %*% beta)))
}
beta0 = lm(Y~Xs[, 2:4])$coefficients
optim(beta0, log_lik, method="BFGS", control=list(fnscale=-1))

## $par
## (Intercept)  Xs[, 2:4]1  Xs[, 2:4]2  Xs[, 2:4]3
##    1.586572    1.372078   -0.007277    0.636273
##
## $value
## [1] -25.62
##
## $counts
## function gradient
##       21       10
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Compared with the results in Part **b**, BFGS algorithm applied on log-likelihood takes less step to converge.

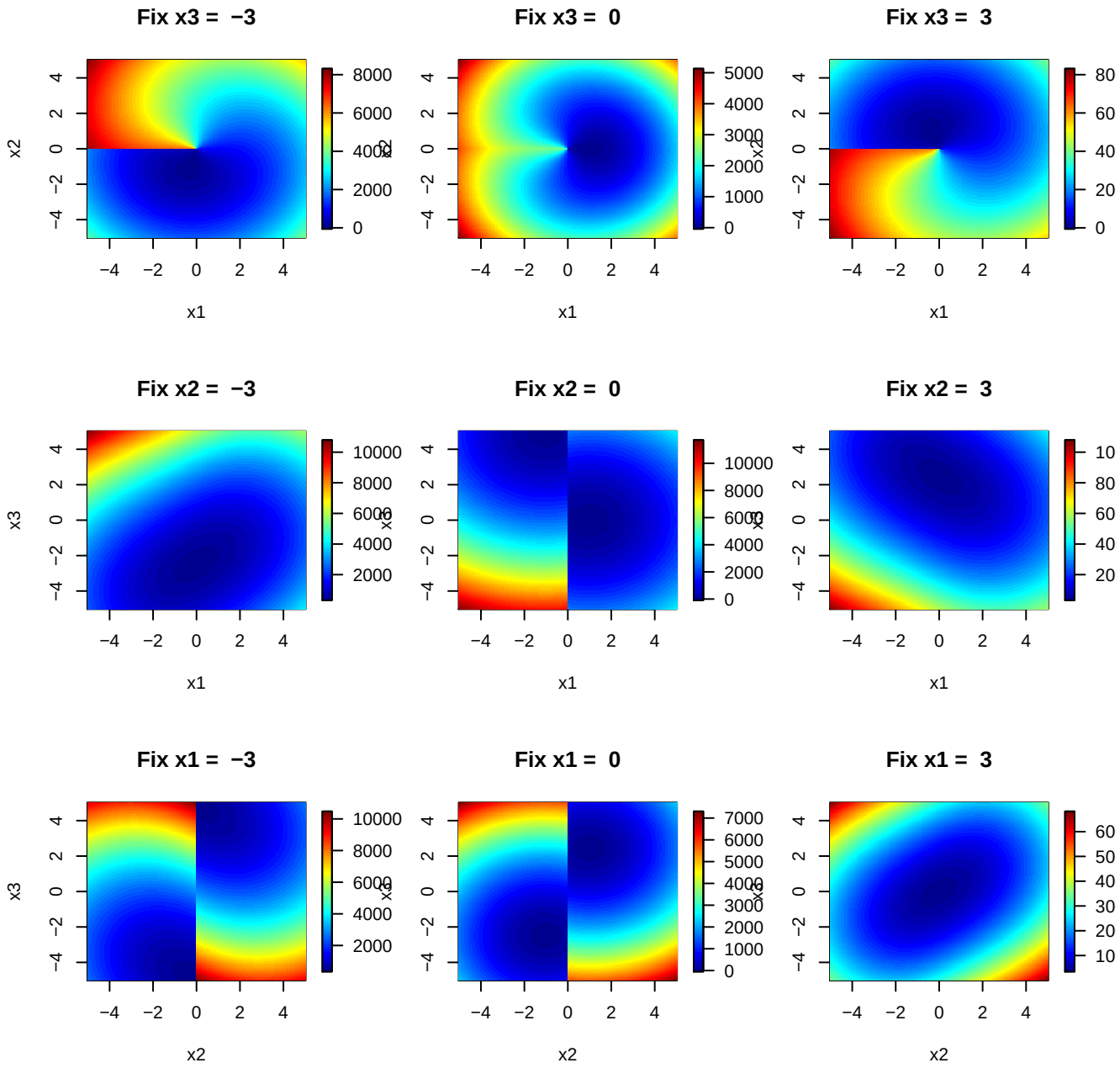
4 Question 4

First of all, I draw slices of the function to get a general idea of the location of the minima.

```
#this file is given
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)
}
# plot it
#install.packages("fields")
#for (x3 in c(-5,0,5))
# image(), contour() persp()
library(fields)

## Loading required package:  spam
## Loading required package:  grid
## Spam version 1.3-0 (2015-10-24) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g.  'help( chol.spam)'.
##
## Attaching package:  'spam'
##
```

```
## The following objects are masked from 'package:base':
##
##    backsolve, forwardsolve
##
## Loading required package:  maps
##
##  # ATTENTION: maps v3.0 has an updated 'world' map.        #
##  # Many country borders and names have changed since 1990.  #
##  # Type '?world' or 'news(package="maps")'.  See README_v3.  #
```

```
par(mfrow=c(3, 3))
n = 200
x1 = seq(-5, 5, length.out = n)
x2 = seq(-5, 5, length.out = n)
x3 = seq(-5, 5, length.out = n)
par(mfrow=c(3, 3))
## fix x1
for (k in c(-3,0,3)){
f_res = apply(expand.grid(x1, x2), 1,
              function(x) {return(f(c(x, k)))})
image.plot(x1, x2, matrix(f_res, ncol = n), main = paste("Fix x3 = ", k)) }
## fix x2
for (k in c(-3,0,3)){
f_res = apply(expand.grid(x1, x3), 1,
              function(x) {return(f(c(x[1], k, x[2])))})
image.plot(x1, x3, matrix(f_res, ncol = n), main = paste("Fix x2 = ", k))
}
## fix x3
for (k in c(-3,0,3)){
f_res = apply(expand.grid(x2, x3), 1,
              function(x) {return(f(c(k, x)))})
image.plot(x2, x3, matrix(f_res, ncol = n), main = paste("Fix x1 = ", k))
}
```



Then, two functions are used for explore the minimum value of the function. In addition, several initial values are also been tried to test the step used for converging. I used *optimx()* function in *optimx package* to compare the result of each method. The results is shown as followed, in which various initial point is choosed.

```
options(digits = 16)
# install.packages("optimx")
require(optimx)

## Loading required package:  optimx

optim(c(1,1,1),f, method = "Nelder-Mead")

## $par
## [1]  0.9999779414172198511 -0.0001349269401560622 -0.0001927127309429892
##
## $value
```

```
## [1] 1.343098332813011e-07
##
## $counts
## function gradient
##      172       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
see = optimx(c(1, 5, 1), f, method = c("Nelder-Mead", "BFGS",  "nlm"))
a = seq(-5,5,length.out =5)
init_set = expand.grid(a,a,a)
#optimize it
minima = numeric(3*nrow(init_set))
for (i in 1:nrow(init_set)) {
  #print((3*(i-1)+1):(3*(i-1)+3))
  minima[(3*(i-1)+1):(3*(i-1)+3)] =
    optimx(as.matrix(init_set)[i,], f, method = c("Nelder-Mead", "BFGS",  "nlm"))$value
}
```

```
## Warning in max(logpar):  no non-missing arguments to max; returning -Inf
## Warning in min(logpar):  no non-missing arguments to min; returning Inf
```

```
summary(minima)
```

```
##                  Min.               1st Qu.                Median
## 0.000000000000000e+00 0.000000000000000e+00 0.000000000000000e+00
##                  Mean               3rd Qu.                  Max.
## 2.398395331961000e+01 9.204955299999999e-06 2.231250000000000e+03
```

```
init_set[as.integer(which(minima>1)/3),]
```

```
##       Var1 Var2 Var3
## 13     0.0  0.0 -5.0
## 38     0.0  0.0 -2.5
## 60     5.0 -2.5  0.0
## 61    -5.0  0.0  0.0
## 63     0.0  0.0  0.0
## 85     5.0 -2.5  2.5
## 88     0.0  0.0  2.5
## 110    5.0 -2.5  5.0
## 111   -5.0  0.0  5.0
## 111.1 -5.0  0.0  5.0
## 112   -2.5  0.0  5.0
## 113    0.0  0.0  5.0
```

```
#see which method give us extreme values
#0:"Nelder-Mead",1: "BFGS",  2: "nlm"
which(minima>.5)%%3
```

```
##  [1] 0 0 2 2 0 2 0 2 0 2 0 0
```

After I select bunch of initial value range from -5 to +5, and store the minima of each method based on those initial values. Almost all the minima are less than 0.001 and (p_1, p_2, p_3) is just around (1,0,0). Therefore I can conclude that 0 is the global minima, and the corresponding coordinate is (1,0,0).

Then let me focus on the extreme value. Two of them don't even converge. These phenomenon happens when x_1, x_3 are large. On the other hand, among three method I tried, only "BFGS" did not slip to extreme values/local minimum, indicating that BFGS method is prefered in this case.