STAT243 Problem set 8

Tianyi Zhang

December 2, 2015

Problem 1

1A

Notice that in this problem part of the explananation for Part B is in Part A. For the purpose of this study, we test the robustness of these two regression methods. A robust method should perform consistently and better with different level of outliers in the sample dataset, and also for various levels of sample sizes.
First, we construct m number of levels of "outliers" datasets with the n levels of different sample sizes. For instance, we could set m=10, and test the datasets with percentage of outliers from 1 to 10. There should be a loop involved, with one loop is from 1 to m, and another loop inside the first loop is indexing from 1 to n.
Secondly, to assess the absolute prediction error. For each particular sample size, we can calculate the sample estimation of the absolute error $\frac{1}{m}\sum_{i=1}^{m}|\hat{Y_i} - Y_i|$ and the squared error $\frac{1}{m}\sum_{i=1}^{m}|\hat{Y_i} - Y_i|^2$, for the coverage of prediction interval, we calculate $\frac{1}{m}\sum_{i=1}^{m}(Number\ of\ new\ observations\ fall\ into\ C_i)$, where C_i is the prediction interval given by the bootstrap calculations. Then we average these measurements over all possible sample sizes. Notice that for the bootstrap step to create prediction interval, we need inputs like the level of prediction interval (like 95 percentage coverage), and a resampling method.

1B

The crucial step for the generation of the simulated dataset is about the generation of certain percentage of outliers. It's hard to control the exact proportion of outliers, thus here by the wikipedia of outliers, outliers can occur in any heavy-tailed distribution or it's actually from another distribution (mixture model). I consider generating outliers from a Gaussian mixture model, and when the outlier level is m, the sample density $P(Y|X)$ comes from:the "main density": $N(X_i^T\theta^{(1)}, \sigma_1^2)$ and the outlier distribution $N(X_i\theta^{(2)}, \sigma_2^2)$. We can change the probability that the sample draws from the outlier distribution to control the different outliers level from 1 to m.

Problem 2

2A

To compare the speed of the decay of the tail for these two distributions, we evaluate the tail function for pareto and exponential distribution: $P(X > x)$.
First we have to make sure that two distributions have the same scale parameters, and we know the scale parameter for pareto is α and the inverse scale parameter for exponential distribution is λ.
Thus, $\lambda = \alpha^{-1}$.**Actually we do not need the same scale assumption to evalute the speed of decay, though this is what I apply here**
For the pareto distribution, when $x > \alpha$, $P(X > x) = (\frac{\alpha}{x})^{\beta}$.

For the exponential distribution, $P(X > x) = \frac{1}{\alpha} e^{\frac{-1}{\alpha} x}$

$$\lim_{x \to \infty} \frac{P_{pareto}(X > x)}{P_{exp}(X > x)} = \frac{(\frac{\alpha}{x})^{\beta}}{\frac{1}{\alpha} e^{\frac{-1}{\alpha} x}}$$

$$\lim_{x \to \infty} \frac{P_{pareto}(X > x)}{P_{exp}(X > x)} = \frac{\alpha^{\beta+1} e^{x/\alpha}}{x^{\beta}}$$

$$By\ l'hopital's\ rule:\ \lim_{x \to \infty} \frac{P_{pareto}(X > x)}{P_{exp}(X > x)} = \infty$$

Thus we conclude that **the Pareto decays slower than the expontential distribution**.

2B

Here I wrote a function called estimation, where it computes $f(x), g(x), h(x)$, and estimates $E(X), E(X^2)$ respectively. Particularlly, I set an option to detect if the user wants a pareto sample density or an exponential sample density so that I do not need to write another function in part 2C. From the histogram, there are not extreme weights in this setting, and the first two moments are close to the theoretical values 3 and 10. Notice that here I use the package VGAM to simulate the data in the pareto distribution.

```
## Create a sample from a Pareto distribution, alpha is the scale in this case
library(VGAM)

## Loading required package:  stats4
##
## Attaching package:  'stats4'
##
## The following object is masked from 'package:spam':
##
##     mle
##
## Loading required package:  splines

Estimation<-function(n,alpha,beta,Paretosample=TRUE){
  if(Paretosample==TRUE){
    sample=rpareto(n,scale=alpha,shape=beta)
   ## Move two units
    fx=dexp(sample-2,rate=1)
    gx=dpareto(sample,scale=alpha,shape=beta)
  }
  else{
      sample=rexp(n,rate=1)+2
      fx=dpareto(sample,scale=alpha,shape=beta)
      gx=dexp(sample-2,rate=1)
  }
  wgts<-fx/gx
  single_emu1<-sample*fx/gx
  single_emu2<-sample*single_emu1
  emu1<-sum(single_emu1)/n
  emu2<-sum(single_emu2)/n
  plotdata<-cbind(wgts,single_emu1,single_emu2)
  return(list(emu1,emu2,plotdata))
}
```

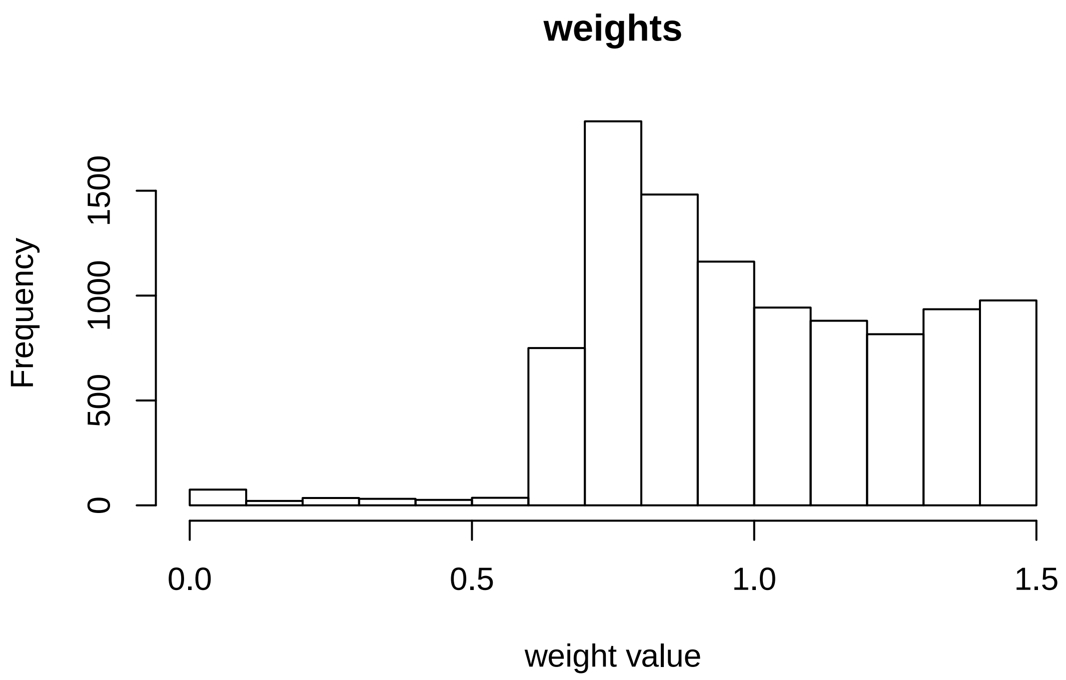
```r
data<-Estimation(10000,alpha=2,beta=3,Paretosample=TRUE)
### EX
data[[1]]
```

```
## [1] 2.997701
```

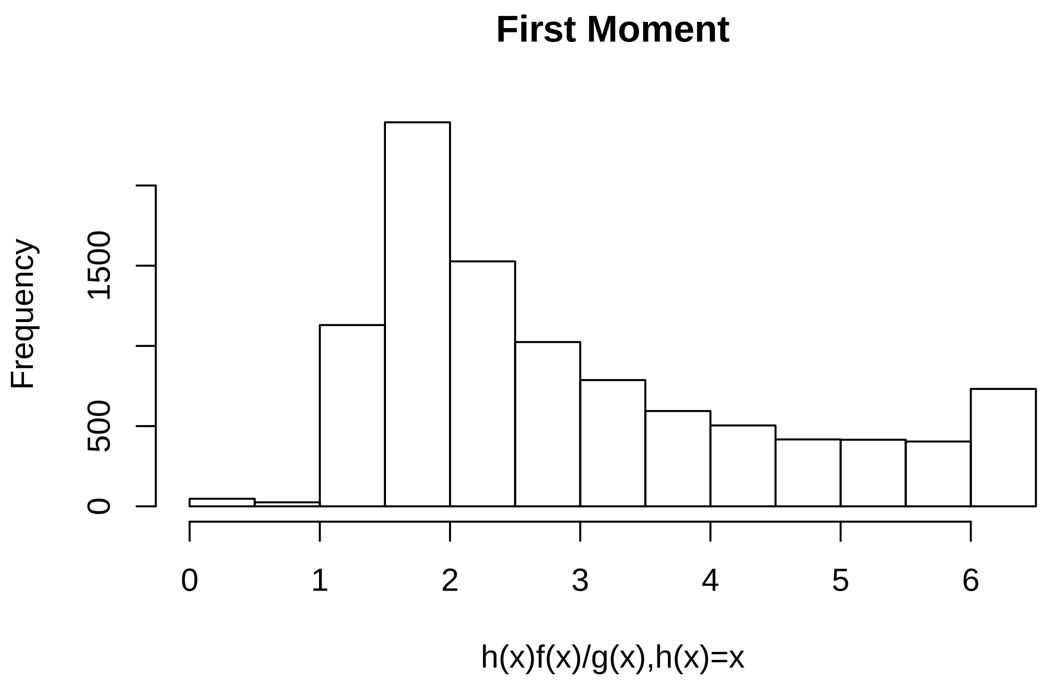
```r
### EX^2
data[[2]]
```

```
## [1] 10.02782
```

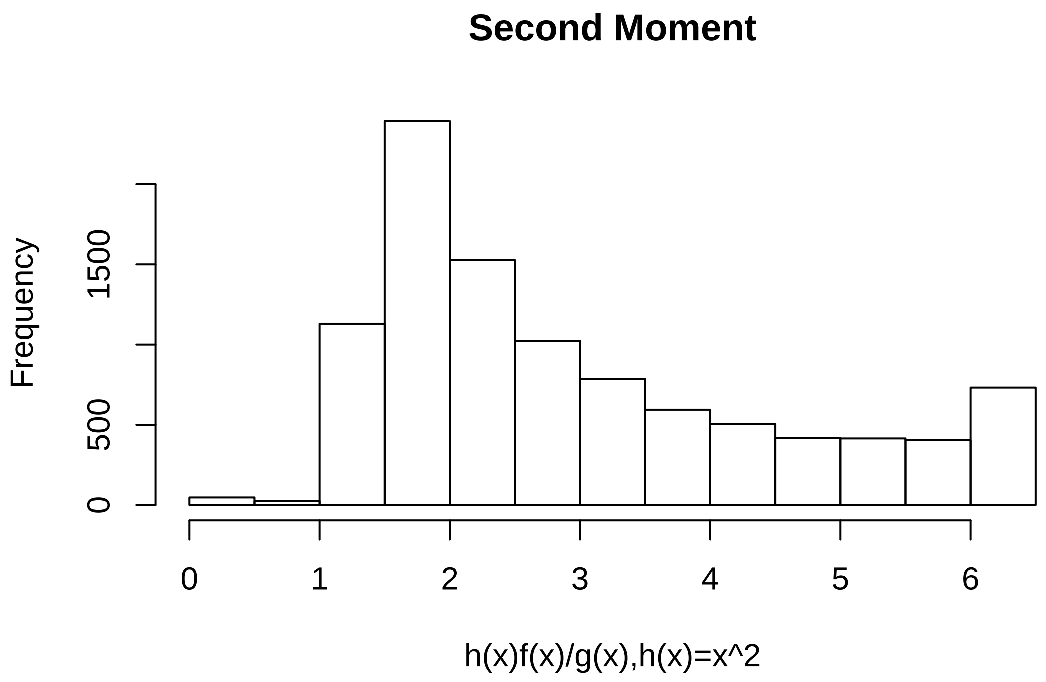
```r
## histogram the weights
## weights
hist(data[[3]][,"wgts"],main="weights",xlab="weight value")
```



```r
## E(X) hf/g
hist(data[[3]][,"single_emu1"],main="First Moment",xlab="h(x)f(x)/g(x),h(x)=x")
```



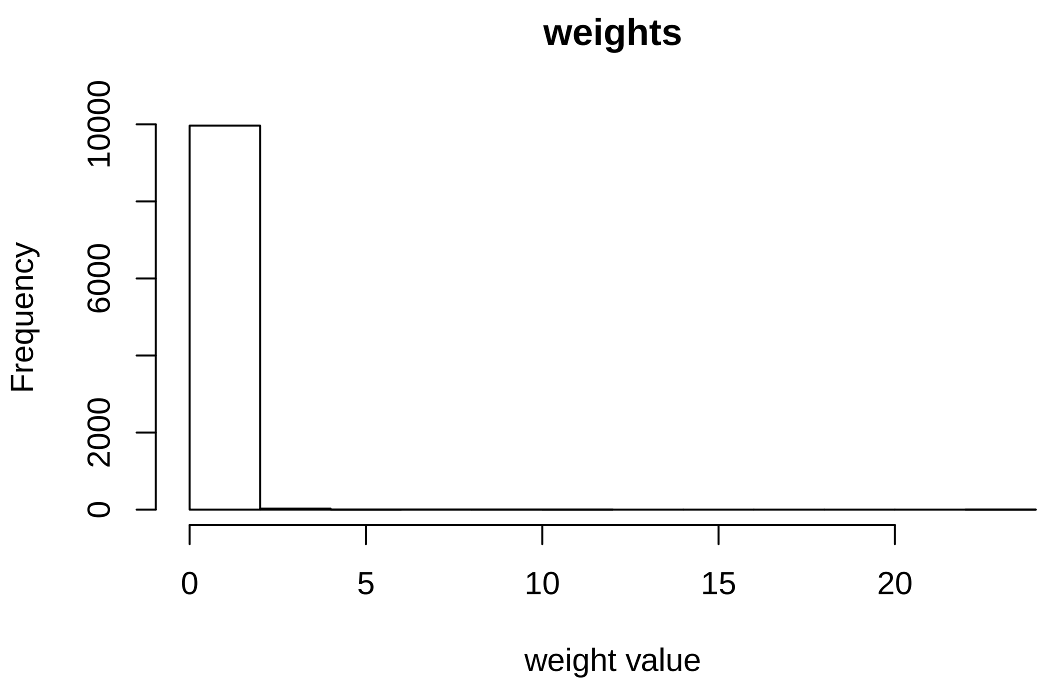
```
hist(data[[3]][,"single_emu1"],main="Second Moment",xlab="h(x)f(x)/g(x),h(x)=x^2")
```



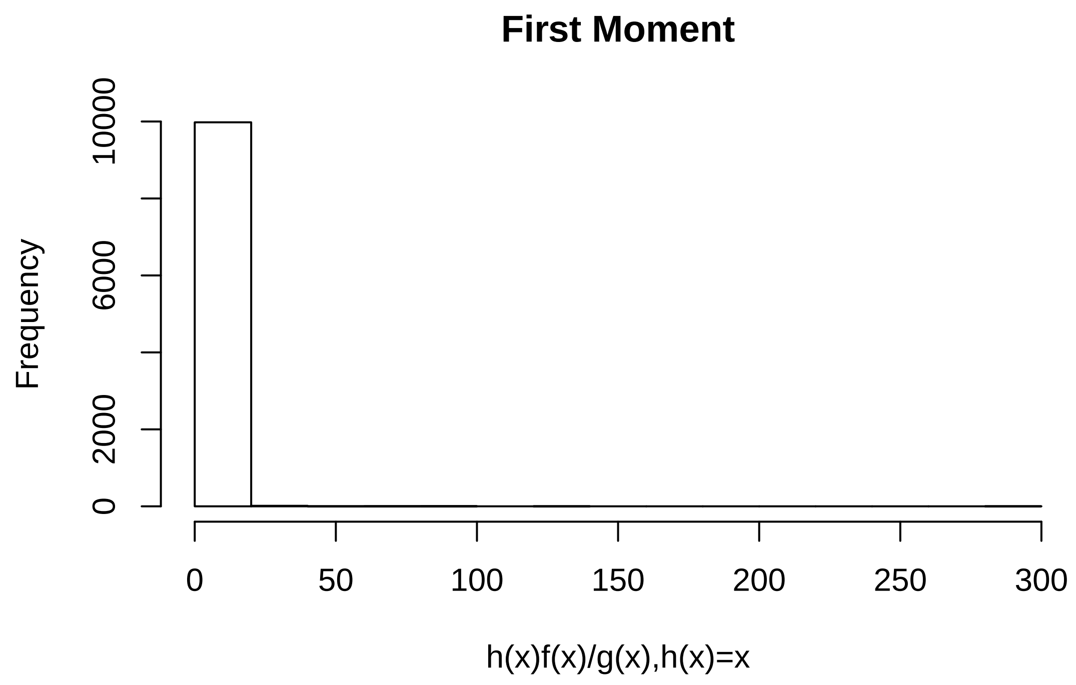
2C

Here in 2C, instead of using a pareto distribution as the sample density, I use the exponential distribution as the sample density. However, in this case, there are several extreme values of weights and $h(x)f(x)/g(x)$ that significantly affect the standard error of the prediction, since $Var(predictor) \propto Var(h(x)f(x)/g(x))$, and this is shown in the histogram. **The variance of the predictors are significantly larger than the method in part B**

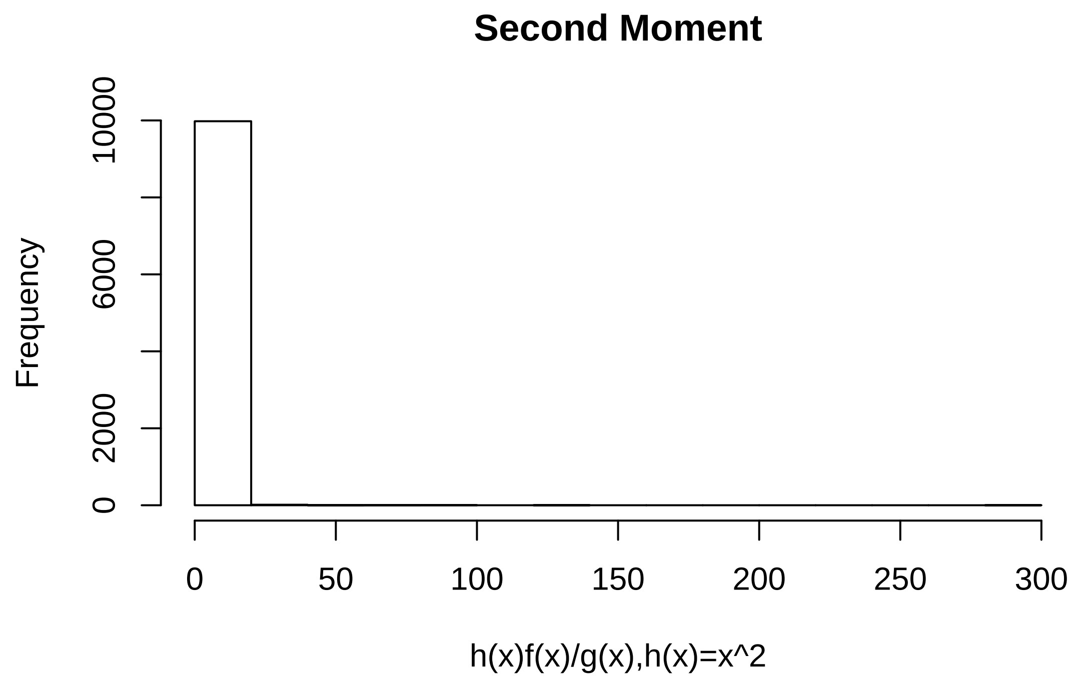
```
data2<-Estimation(10000,alpha=2,beta=3,Paretosample=FALSE)
data2[[1]]

## [1] 2.947173

### EX^2
data2[[2]]

## [1] 10.30246

## histogram the weights
## weights
hist(data2[[3]][,"wgts"],main="weights",xlab="weight value")
```



```
## E(X) hf/g
hist(data2[[3]][,"single_emu1"],main="First Moment",xlab="h(x)f(x)/g(x),h(x)=x")
```



```
hist(data2[[3]][,"single_emu1"],main="Second Moment",xlab="h(x)f(x)/g(x),h(x)=x^2")
```



```r
summary(data2[[3]][,"wgts"])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6927  0.7630  0.9356  1.0010  1.1910 23.8900
```

Problem 3

3A

For this EM algorithm, we are maximizing the function $Q(\beta|\beta^{(t)})$, which is the conditional expectation of the log likelihood function given $Y, X, \beta^{(t)}$.
We know the Likelihood function, by iid property the law of conditional probability:
$L(\beta; Y, Z) = P(Y, Z|\beta) = \Pi p(Z_i, Y_i|\beta) = \Pi p(Z_i|\beta)p(Y_i|Z_i, \beta)$
We know that: $p(Y_i|Z_i, \beta) = I(Z_i > 0)^{Y_i} I(Z_i <= 0)^{1-Y_i} = 1$
We take the log of the likelihood function to get the loglihood function:

$$\begin{aligned} loglihood(\beta|Y, Z) &= \sum((logp(Y_i|Z_i, \beta) + logP(Z_i|\beta))) \\ &= \sum(-\frac{1}{2}log(2\pi) - \frac{1}{2}(Z_i - X_i^T\beta)^2) \\ Q(\beta|\beta^{(t)}) = E[loglihood(\beta|Y, Z)|Y, X, \beta^{(t)}] &= \sum_i E[-\frac{1}{2}log(2\pi) - \frac{1}{2}(Z_i - X_i^T\beta)^2|Y, X, \beta^{(t)}] \\ &= \sum -\frac{1}{2}log(2\pi) - \frac{1}{2}(E[Z_i^2|Y, X, \beta^{(t)} - 2X_i^T\beta E[Z_i|Y, X, \beta^{(t)}] + (X_i^T\beta)^2) \end{aligned}$$

Unsuprisingly, the problem has been transformed to another ordinary regression problem of new quantities:
Maximizing $Q(\beta|\beta^{(t)})$ is equivalent to minimizing $\sum_i -2X_i^T\beta E[Z_i|Y, X, \beta^{(t)}] + (X_i^T\beta)^2)$ over β
Thus, from the equation of ordinary least squares regression:
$\hat{\beta}^{(t+1)} = (X^TX^{-1}X^TE[Z|Y, X, \beta^{(t)}])$
Where the equation of $E[Z|Y, X, \beta_t]$ is given in the hints section, when $Y_i = 1, Z_i > 0, E = X_i^T\beta^{(t)} + \frac{\phi(X_i^T\beta^{(t)})}{\Phi(X_i^T\beta^{(t)})}$

3B

As discussed in class, the starting value of the estimator could be the estimator where we ignore the latent variables Z, such that:
$\beta^{(0)} = (X^TX)^{-1}XY$

3C

First we have to simulate the dataset with sample size of 100, with dimension of the beta vector equals to 3, and there is also an intercept β_0. The true β_0 and β_1 I set is -1 and 2, where β_0 denotes the intercept.

```r
library(Rlab)

## Rlab 2.15.1 attached.
##
##
## Attaching package:  'Rlab'
##
## The following objects are masked from 'package:fields':
##
##    bplot, bplot.xy, cat.to.list, describe, set.panel, stats, US,
##     world, xline, yline
##
## The following object is masked from 'package:maps':
##
##     ozone
##
## The following objects are masked from 'package:stats':
##
##     dexp, dgamma, dweibull, pexp, pgamma, pweibull, qexp, qgamma,
##     qweibull, rexp, rgamma, rweibull
##
## The following object is masked from 'package:datasets':
##
##     precip
```

```
set.seed(0)
n=100
beta=c(-1,2,0,0)
X=cbind(rep(1,n),matrix(rnorm(n*(length(beta)-1)),ncol=length(beta)-1))
## Probability Y=1 (cumulative probabiity function)
py1<-pnorm(X %*% beta)
Y<-sapply(py1,function(x){return(rbern(1,x))})

## expressing that expectation
eachstep<-function(X,Y,beta){
  Xbeta<-X %*% beta
  Expectation=rep(0,length(Y))
  for(i in 1:length(Y)){
    Expectation[i]<-ifelse(Y[i]==1,Xbeta[i]+dnorm(Xbeta[i])/pnorm(Xbeta[i]),Xbeta[i]-dnorm(Xbeta[i])/pnc
  }
  ord_reg<-lm(Expectation~X[,2:4])$coefficients
  return(ord_reg)
}
```

Here I write the major function for this probit model, where I set the criteria to stop the iteration process where the iteration stops at either the max absoulte error is less than 10^{-6} or the number of iterations has reached 1000. Finally when I run my program, the number of iterations are about 224 times.

```
probit_new<-function(X,Y,error=1e-06,term_cond=1000){
  beta_old<-lm(Y~X[,2:4])$coefficients
  beta_new<-eachstep(X,Y,beta_old)
  num_itr<-1
  while(max(abs(beta_new-beta_old))>error & num_itr<term_cond){
    beta_old<-beta_new
    beta_new<-eachstep(X,Y,beta_old)
    num_itr<-num_itr+1
  }
  print(c("number of iterations=",num_itr))
  names(beta_new)<-c("beta0","beta1","beta2","beta3")
  return(beta_new)
}
probit_new(X,Y)

## [1] "number of iterations=" "224"
##      beta0      beta1      beta2      beta3
## -1.2880168  2.3026992  0.1942564 -0.1237856
```

3D

Y_i follows a Bernoulli distribution with $Ber(\Phi(X_i^T\beta))$, thus:

$$p(Y|X,\beta) = \prod_i \Phi(X_i^T\beta)^{Y_i}(1-\Phi(X_i^T\beta))^{1-Y_i}$$

Thus, the loglihood function is:

$$l(\beta|Y,X) = \sum_i [Y_i \log(\Phi(X_i^T\beta)) + (1-Y_i)\log(1-\Phi(X_i^T\beta))]$$

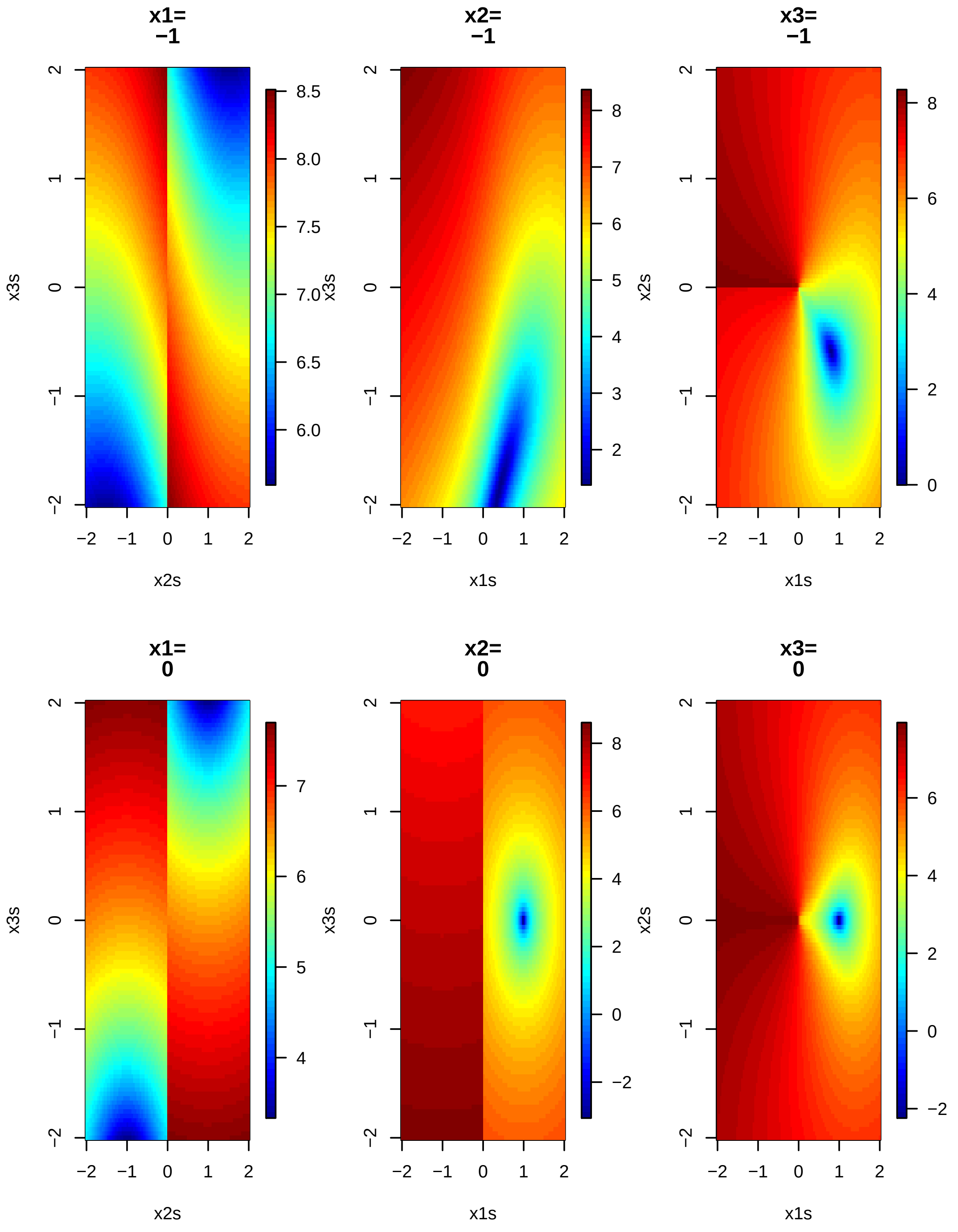
The result for beta estimation and SE I got was the same if I use the optim or the built in glm option in R. These results are also consistent with the one when I was using the EM Algorithm. The number of iterations are 46 for the BFGS option.

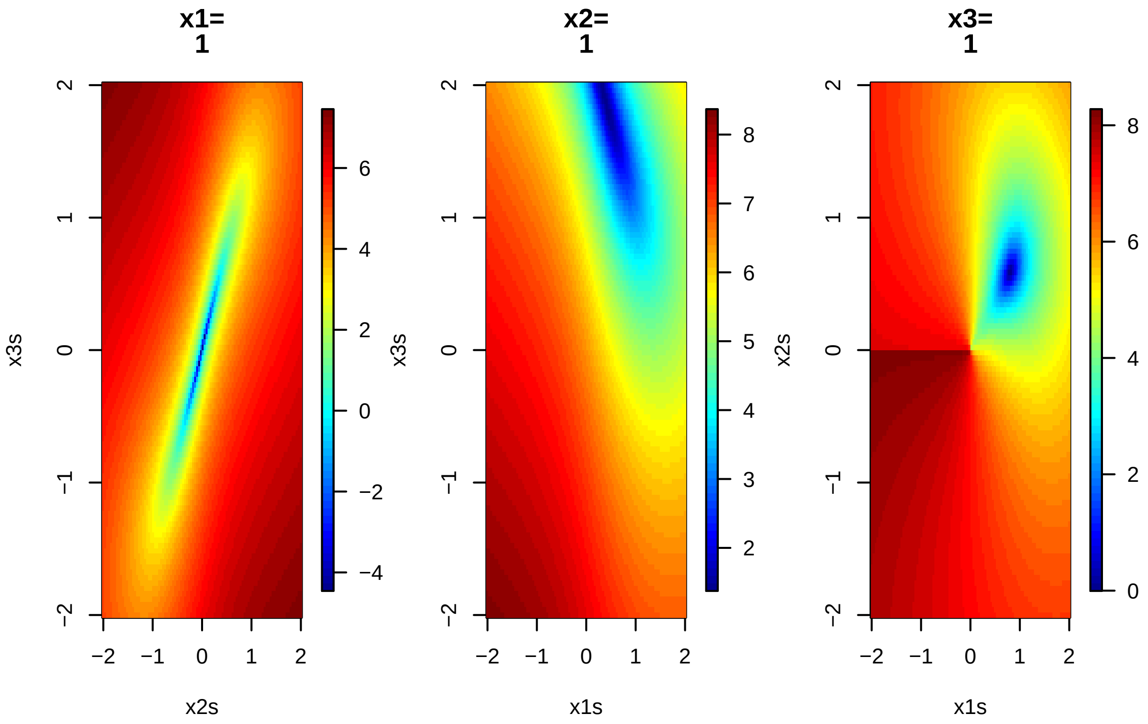
```
loglihood<-function(X,Y,inits){
  Xbeta=X%*%inits
  logli=sum(Y*log(pnorm(Xbeta))+(1-Y)*(log(1-pnorm(Xbeta))))
  return(logli)
}
inits<-lm(Y~X[,2:4])$coefficients
result<-optim(inits,loglihood,X=X,Y=Y,method="BFGS",control=list(fnscale=-1),hessian=TRUE)

x1=X[,2]
x2=X[,3]
x3=X[,4]
formal_result<-glm(Y~x1+x2+x3,family=binomial(link="probit"))
## iterations
result$counts

## function gradient
##       46       13

## Beta
result$par

## (Intercept)   X[, 2:4]1   X[, 2:4]2   X[, 2:4]3
##  -1.2880166   2.3027679   0.1940882  -0.1239512

formal_result$coefficients

## (Intercept)          x1          x2          x3
##  -1.2880267   2.3027164   0.1942575  -0.1237867

##SE
se<-sqrt(-diag(solve(result$hessian)))
se_formal<-coef(summary(formal_result))[,"Std. Error"]
se

## (Intercept)   X[, 2:4]1   X[, 2:4]2   X[, 2:4]3
##   0.2865534   0.4524275   0.2229270   0.2083821

se_formal

## (Intercept)          x1          x2          x3
##   0.2818166   0.4395725   0.2163372   0.2051311
```

Problem 4

In this problem, I first try to plot silces to see the behaviors of the function f, where I fix the value of x_1, x_2, x_3 respectively, and gave each of the input value from -1 to 1. As it's shown on the plot, it seems that the choice $(1, 0, 0)$ gives the smallest value. Besides, as shown on the graph, many plots are symmetric or half symmetric, thus multiple local minimums are possible by direct inspection, however it's not the case by further inspection.

```r
## Another
library("fields")
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)
}
x1s <- seq(-2, 2, len = 100); x2s = seq(-2, 2, len = 100); x3s =seq(-2,2,len=100)
## Fixing one input and change others
par(mfrow=c(1,3))
for (i in c(-1,0,1)){
  ## Fix x1
  fx1=apply(expand.grid(x2s, x3s), 1,function(x){return(f(c(i,x)))})
  image.plot(x2s, x3s, matrix(log(fx1),100,100), main=c("x1=",i))
  ## Fix x2
  fx2=apply(expand.grid(x1s, x3s), 1,function(x){return(f(c(x[1],i,x[2])))})
  image.plot(x1s, x3s, matrix(log(fx2),100,100), main=c("x2=",i))
  ## Fix x3
  fx3=apply(expand.grid(x1s, x2s), 1,function(x){return(f(c(x,i)))})
  image.plot(x1s, x2s, matrix(log(fx3),100,100), main=c("x3=",i))
}
```


x1= −1
x2= −1
x3= −1
x1= 0
x2= 0
x3= 0
x2s
x1s
x3s
x2s
8.5
8.0
7.5
7.0
6.5
6.0
−2
−1
0
1
2



Here I explore the global minimum of the function f, and as it's shown on the previous plots, the possible global minimum is close to $(1, 0, 0)$. Thus we would like to start from $(0, 0, 0), (10, 10, 10), (-10, -10, -10)$ and search for the global minimum using a variety of optimization methods in optimx, but here I just use the Nelder-Mead method in optim to illustrate. The convergence zero shows that the optimization has been completed succesfully, and it's clear that whatever the starting point is, the local minmimum returned is $x = (1, 0, 0)$, thus there are no mutiple local minimums.
Besides, it is super obvious that $x = (1, 0, 0)$ reaches the global minimum, because the f returns the result with the nonnegative value $f_1^2 + f_2^2 + f_3^2$ and $f(1, 0, 0) = 0$.

```
init1 <- c(0,0,0)
optim(init1, f, method = "Nelder-Mead")

## $par
## [1] 0.999978292 0.002730698 0.004284640
##
## $value
## [1] 1.876851e-05
##
## $counts
## function gradient
##      110       NA
##
## $convergence
## [1] 0
##
## $message
## NULL

init2<- c(10,10,10)
optim(init2, f, method = "Nelder-Mead")
```

```
## $par
## [1]  1.000050952 -0.009061762 -0.014139314
##
## $value
## [1] 0.0002087078
##
## $counts
## function gradient
##      252       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
init3<-c(-10,-10,-10)
optim(init3, f, method = "Nelder-Mead")
```

```
## $par
## [1] 1.000019191 0.001776860 0.003591942
##
## $value
## [1] 7.132079e-05
##
## $counts
## function gradient
##      258       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
f(c(1,0,0))
```

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