PS8

Mengying Yang

December 4, 2017

Problem 1

*a)

To know if Pareto decay more quickly or not than exponantial distribution, we have $exp(x) = \lambda e^{-\lambda x}$ and $fPareto(x) = \frac{\beta\alpha^\beta}{x^{\beta+1}} I\{\alpha < x\}$. Therefore

$\lim_{x\to\infty} \frac{exp(x)}{fPareto(x)} = \lim_{x\to\infty} \frac{\lambda e^{-\lambda x}}{\frac{\beta\alpha^\beta}{x^{\beta+1}}}$

As we can see numerator and denominator are 0 when x goes to infinity, so we can apply the L'Hospital's rule. We need to do is differentiate the numerator and differentiate the denominator and then take the limit. We can find the numerator() is a constant and the denominator goes to infinity,

$\lim_{x\to\infty} \frac{exp(x)}{fPareto(x)} = \lim_{x\to\infty} \frac{\lambda}{\lambda^{\beta+1}\beta\alpha^\beta e^{\lambda x}} = 0$

Thus, the tail of the Pareto decay more slowly than that of an exponential distribution.

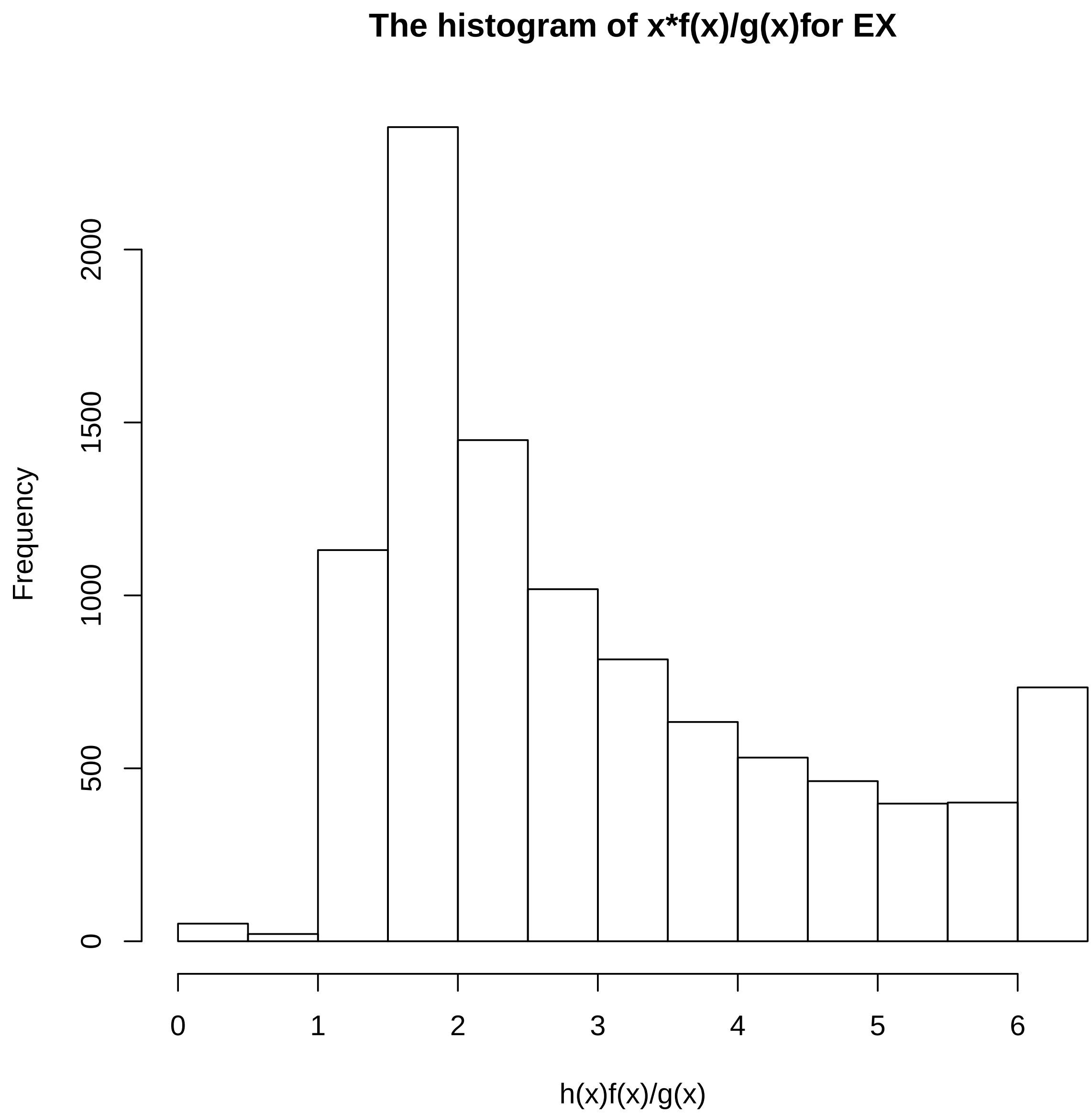
Problem1

b)

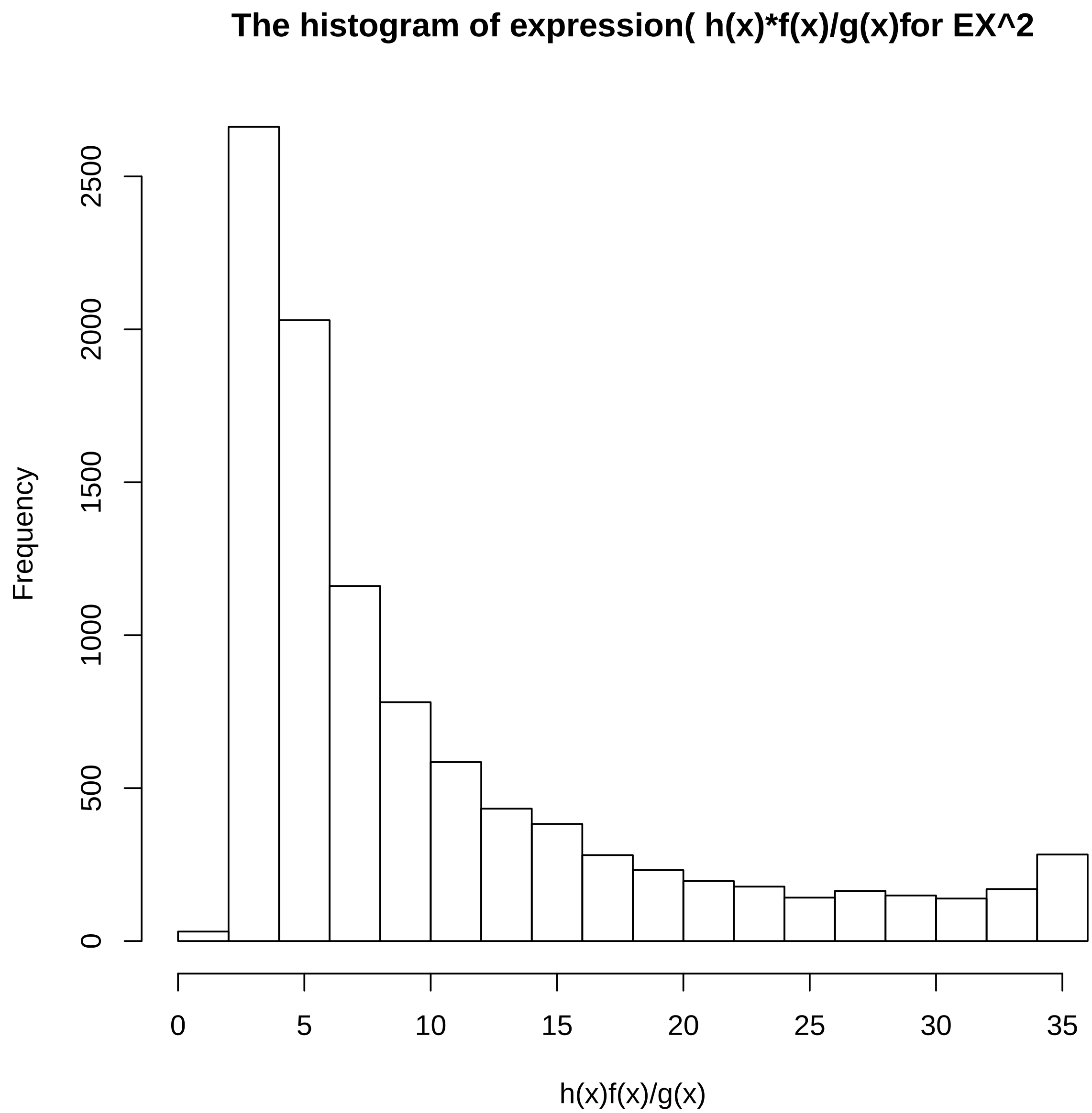
```
library(EnvStats)

##
## Attaching package:  'EnvStats'
## The following objects are masked from 'package:stats':
##
##     predict, predict.lm
## The following object is masked from 'package:base':
##
##     print.default

#Estimate E(X) and E(X^2) when m = 10000
ParetoSample <- rpareto(10000,location = 2, shape = 3)
weight <- dexp(ParetoSample-2)/dpareto(ParetoSample,location =2,shape =3)
X <- weight*ParetoSample
#Expectation of X, should close to 3
mean(X)

## [1] 3.022294

X_Square <- weight*ParetoSample^2
##Expectation of X_Square, should close to 10
mean(X_Square)

## [1] 10.08577
```

For exponetial distibution with parameter =1 and shift by 2. E(X) = 1 + 2 =3 and Variance remain the same. Therefore, the $E(X^2) = Var(X) + E(X^2) = 1 + 9 = 10$

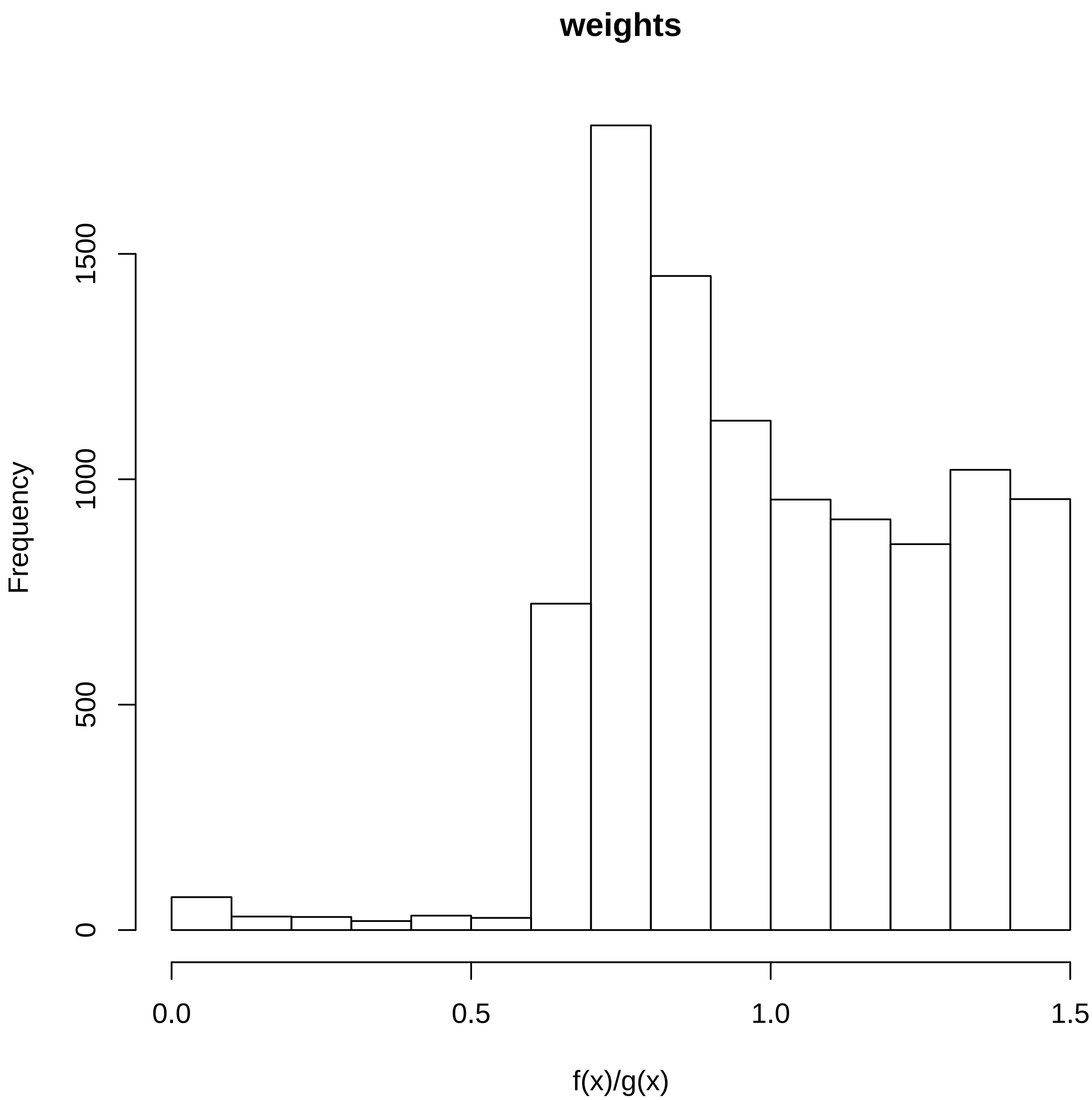
```
# Histogram of h(x)f(x)/g(x)
hist(X, xlab = "h(x)f(x)/g(x)", main="The histogram of x*f(x)/g(x)for EX")
```




```
hist(X_Square, xlab = "h(x)f(x)/g(x)", main="The histogram of expression( h(x)*f(x)/g(x)for EX^2")
```




```
#the weight f(x)/g(x)
hist(weight, xlab = "f(x)/g(x)", main= "weights")
```

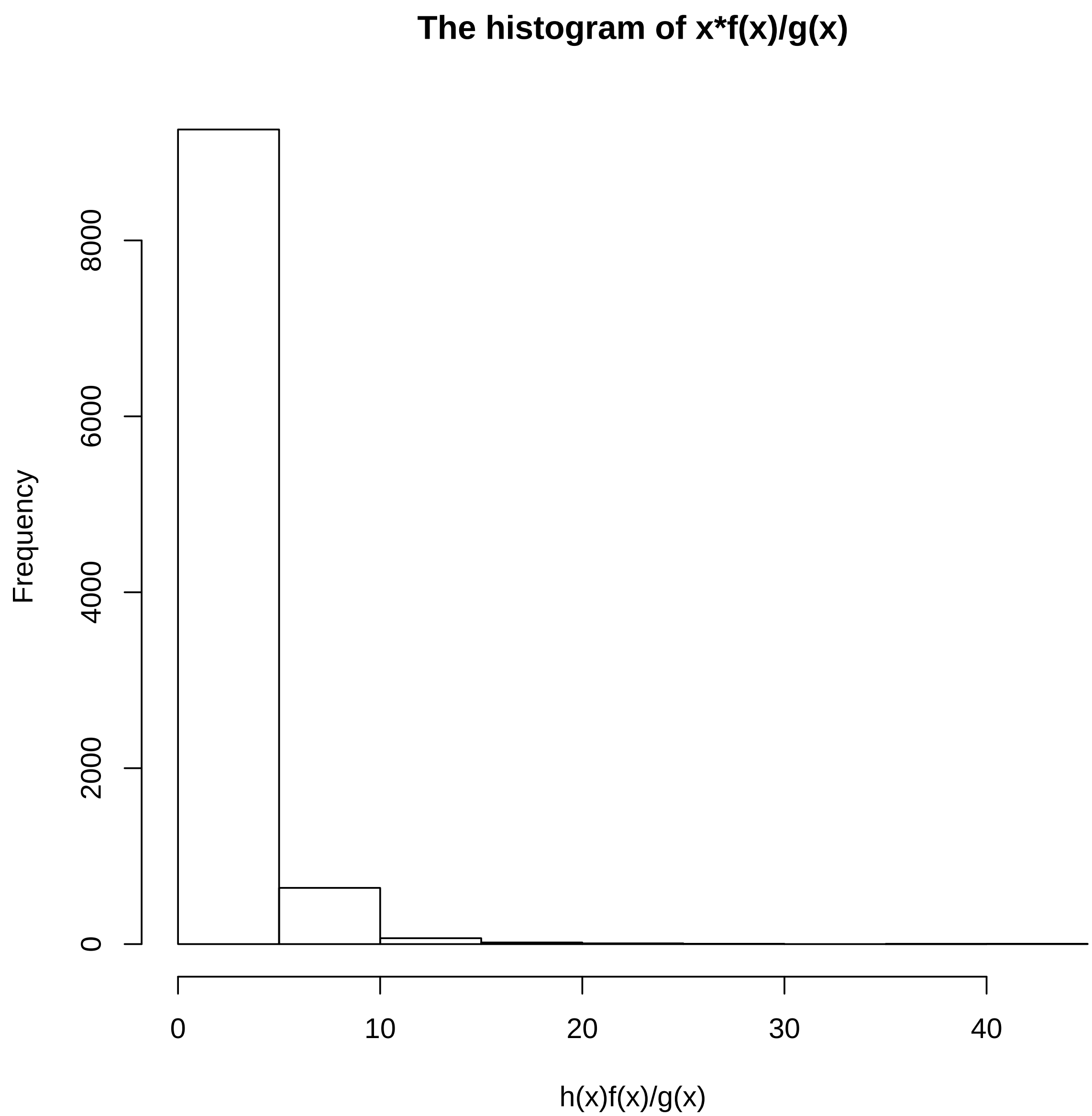


As we can see from the histogram, the weights lie in the interval (0.5,1.5) and rest few lie in (0,0.5). Therefore, there is no strong influence on the estimated h(X)

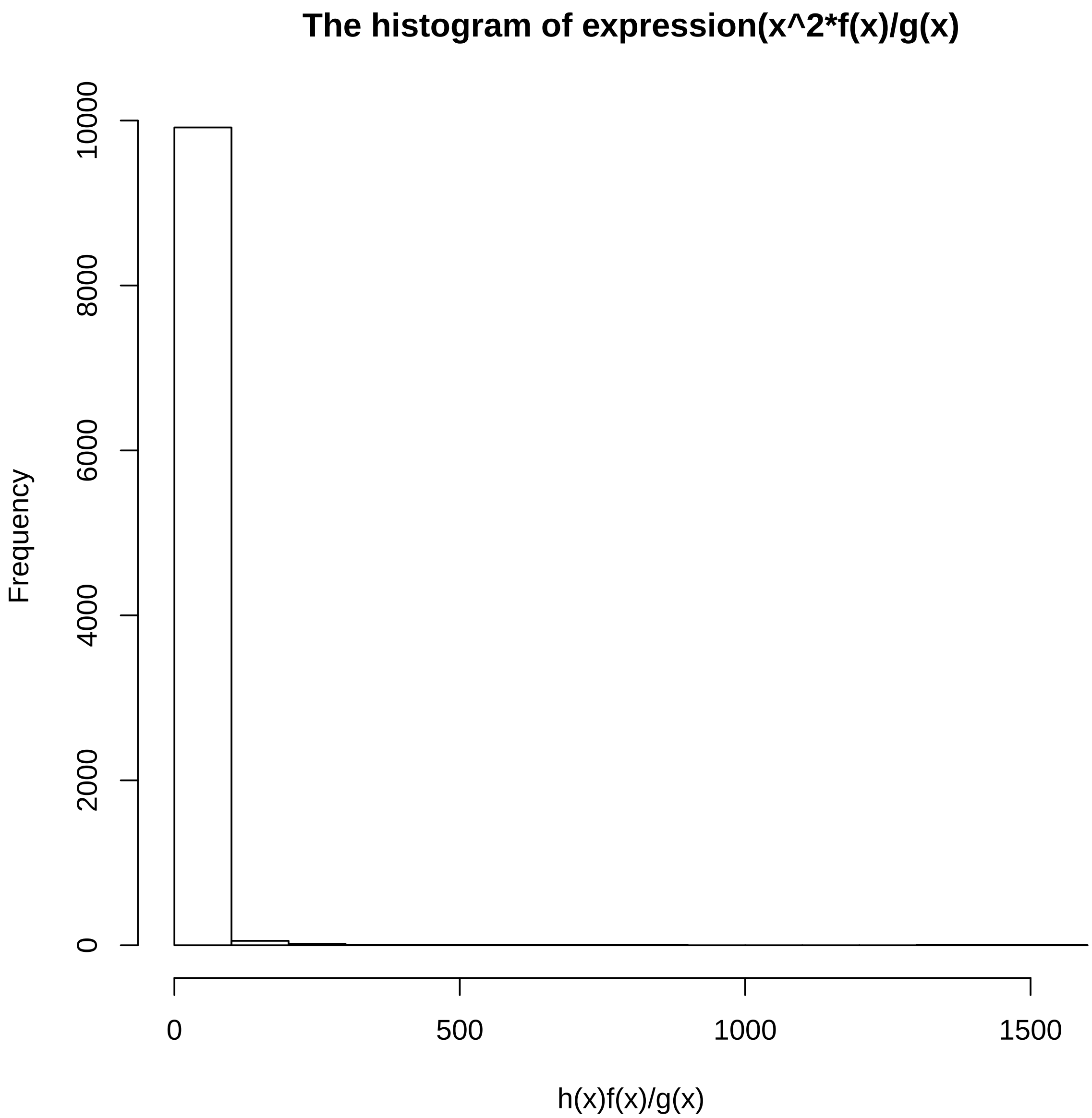
c)

```
#we exchange the f and g
#Estimate E(X) and E(X^2) when m = 10000
ExponetialSample <- rexp(10000) + 2
weight_new <- dpareto(ExponetialSample,location =2,shape =3)/dexp(ExponetialSample-2)
X_new <- weight_new*ParetoSample
#Expectation of X, should close to 3
mean(X_new)

## [1] 2.981714
```

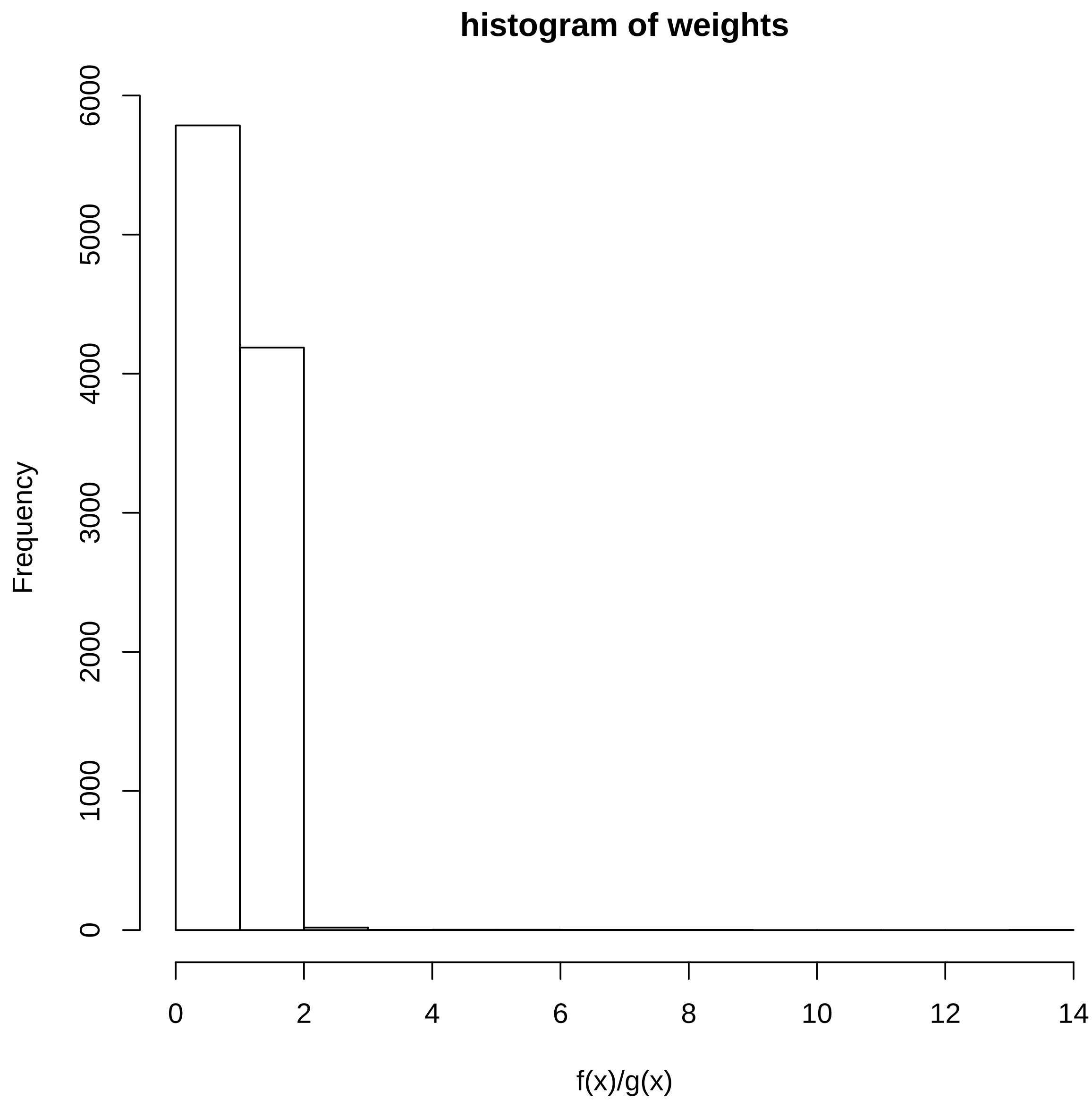
```
X_Square_new <- weight_new*ParetoSample^2
##Expectation of X_Square, should close to 10
mean(X_Square_new)

## [1] 11.67727

# Histogram of h(x)f(x)/g(x)
hist(X_new, xlab = "h(x)f(x)/g(x)", main="The histogram of x*f(x)/g(x)")
```




```
hist(X_Square_new, xlab = "h(x)f(x)/g(x)", main="The histogram of expression(x^2*f(x)/g(x)")
```




```
#the weight f(x)/g(x)
hist(weight_new, xlab = "f(x)/g(x)", main= "histogram of weights")
```

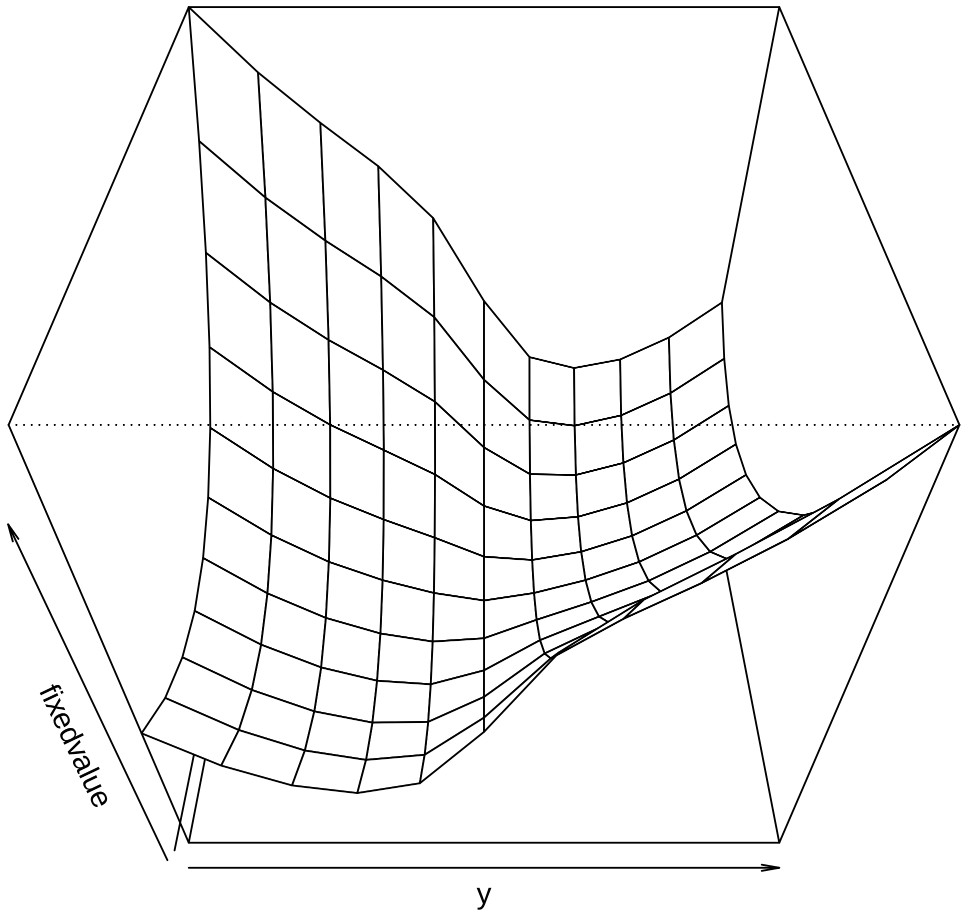


From the histogram we can find the very strong variance of the weights and it influences the estimated h(X) indirectly. Because the exponential distribution has fast decaying tail(lighter tail), the variance becomes large when we sampling from exponetial distribution.

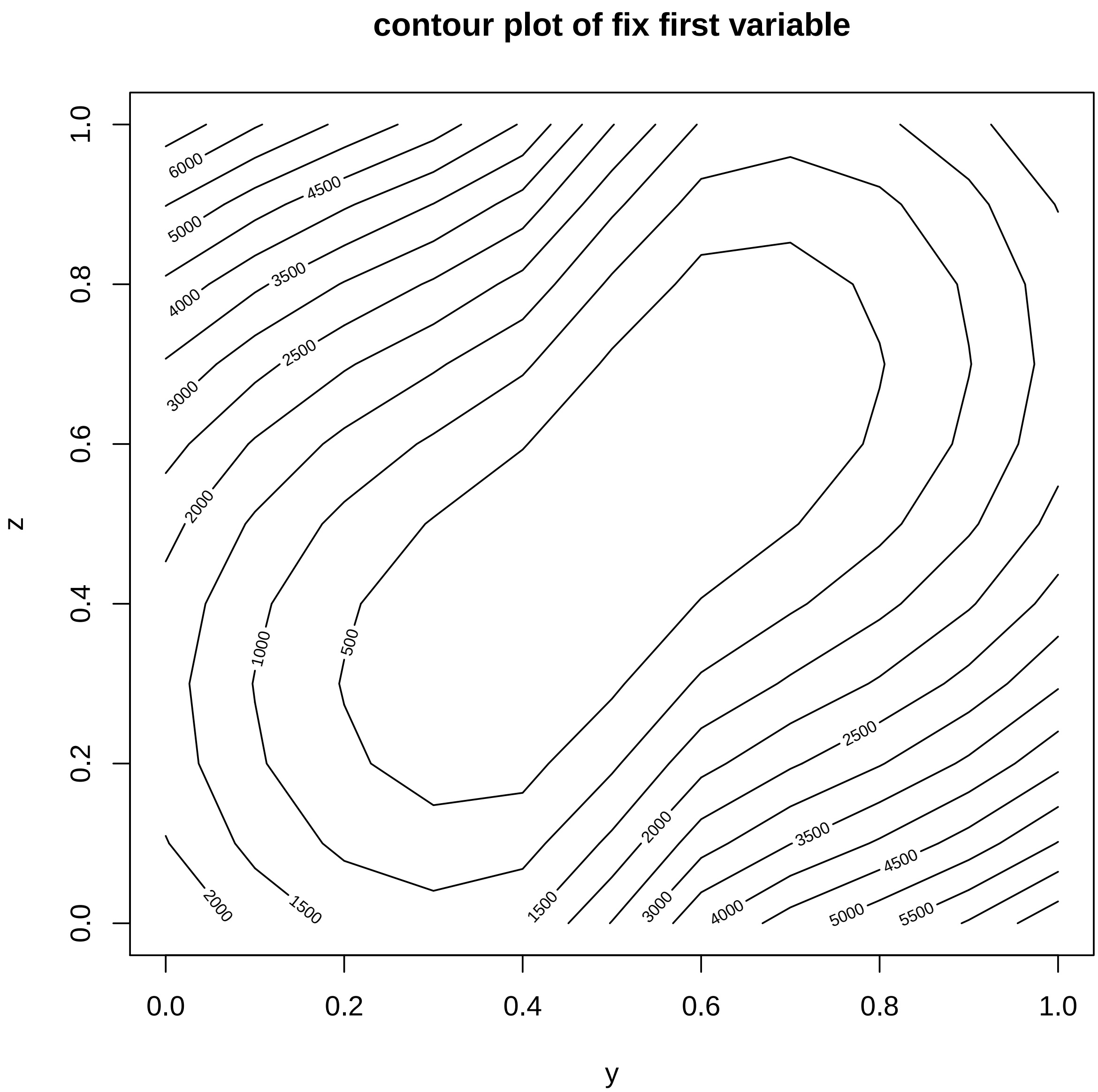
Problem 2

```
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)
```

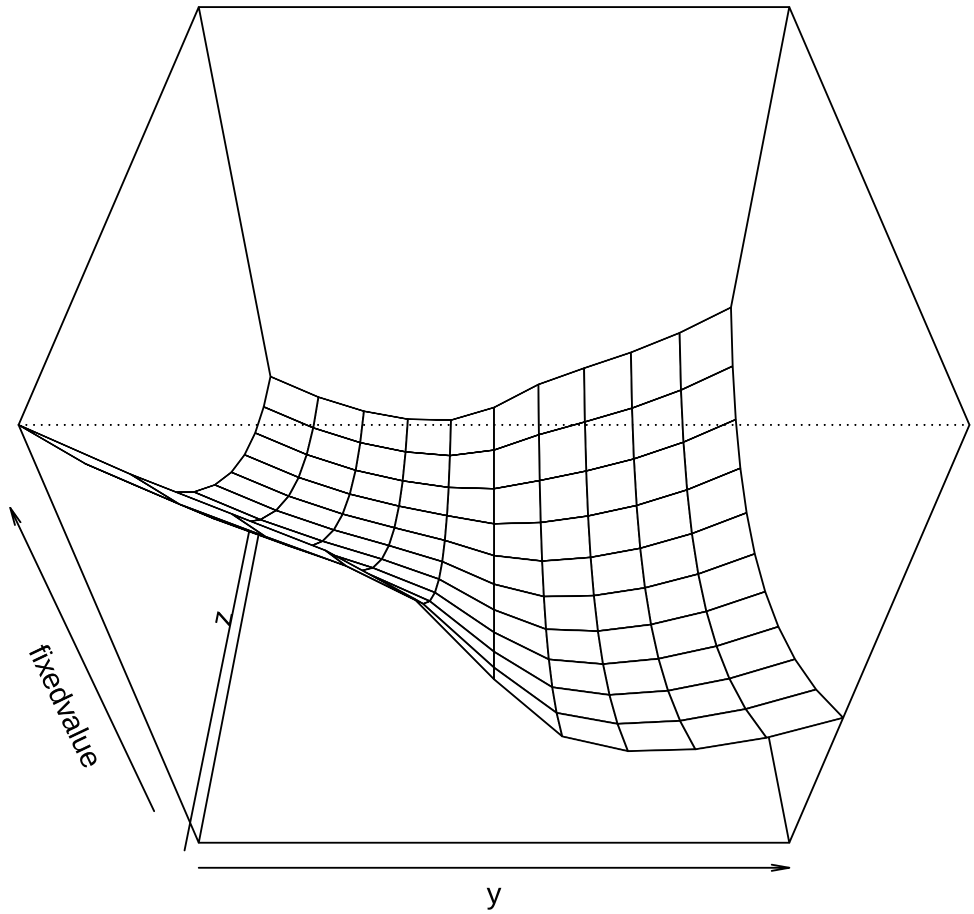
```
}
# fix one variable, the other two range from -5 to 5
# fixed the first variable as constant
Fixed_1 <- function(y,z){
  return(f(c(1,y,z)))
}
#to make the Fixed_1 able to calculate, we need to use vecterize
vectorized <- Vectorize(Fixed_1 ,vectorize.args = c("y","z"))
matrix_1<- outer(-5:5, -5:5, vectorized)

#dataframe_1 <- data.frame(expand.grid(x=-5:5, y=-5:5),value = c(t(matrix_1)))

#plots of a surface over the y-z plane
persp(matrix_1, phi = 45, zlab = "fixedvalue", xlab ="y", ylab = "z")
```



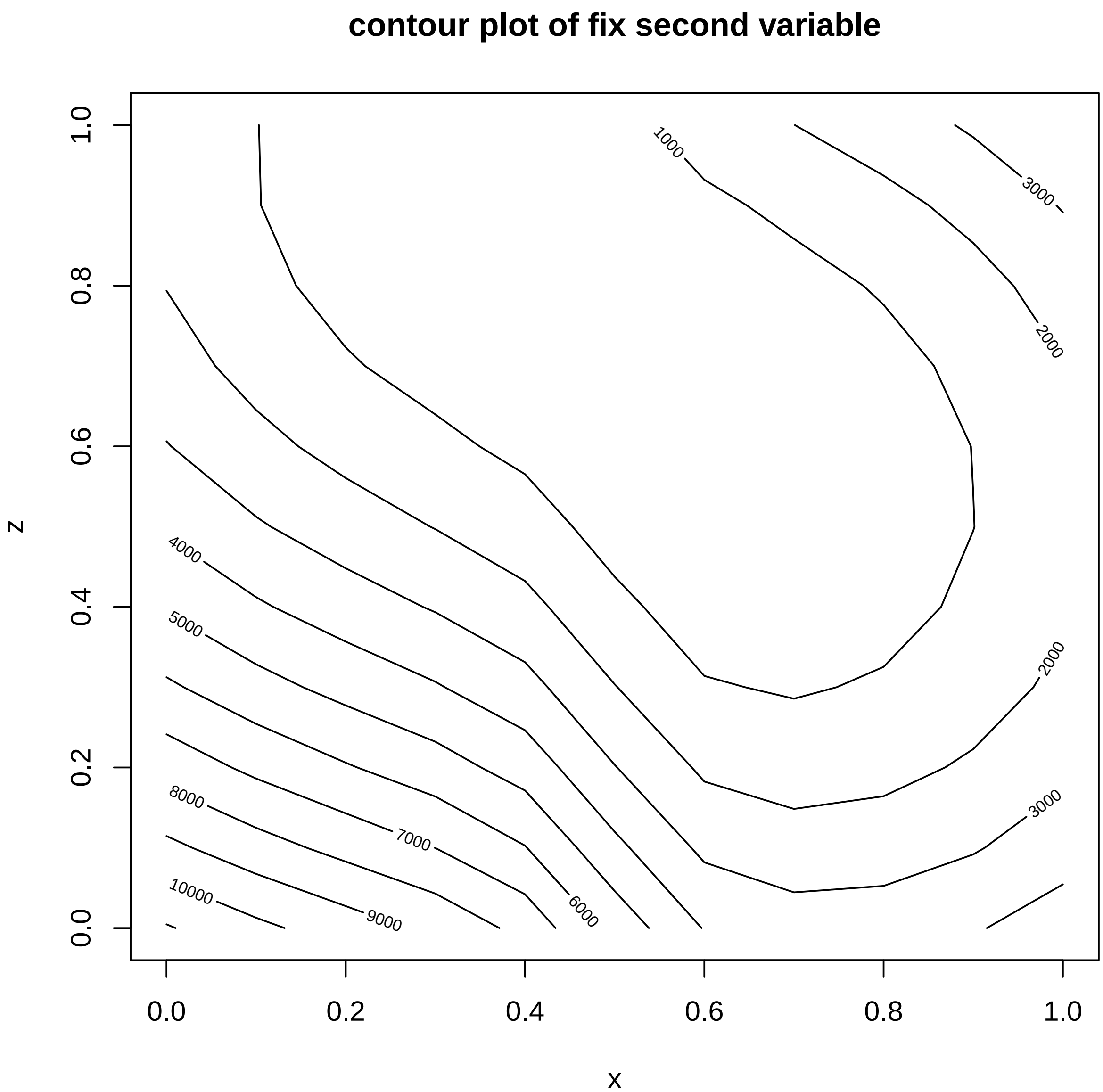

```r
# making a contour plot
contour(matrix_1,  xlab ="y", ylab = "z", main ="contour plot of fix first variable")
```



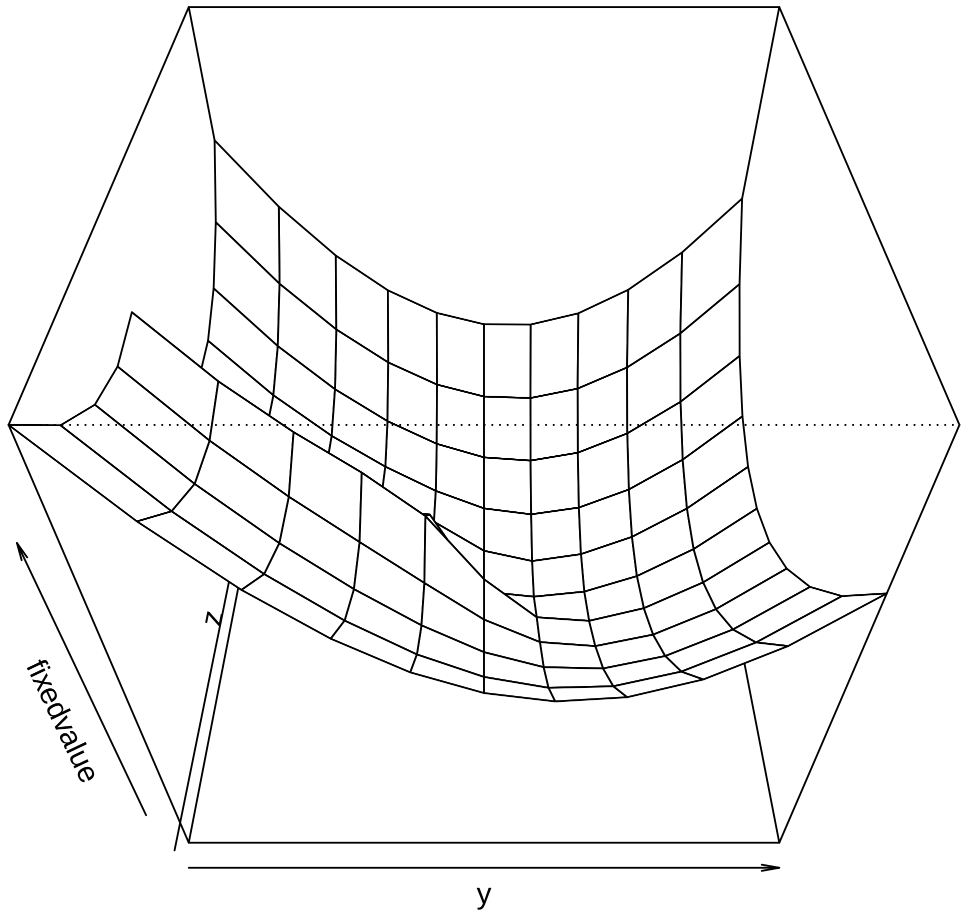

```r
# fixed the second variable as constant
Fixed_2 <- function(x,z){
  return(f(c(x,1,z)))
}
#to make the Fixed_1 able to calculate, we need to use vecterize
vectorized <- Vectorize(Fixed_2 ,vectorize.args = c("x","z"))
matrix_2<- outer(-5:5, -5:5, vectorized)

#plots of a surface over the y-z plane
persp(matrix_2, phi = 45, zlab = "fixedvalue", xlab ="y", ylab = "z")
```



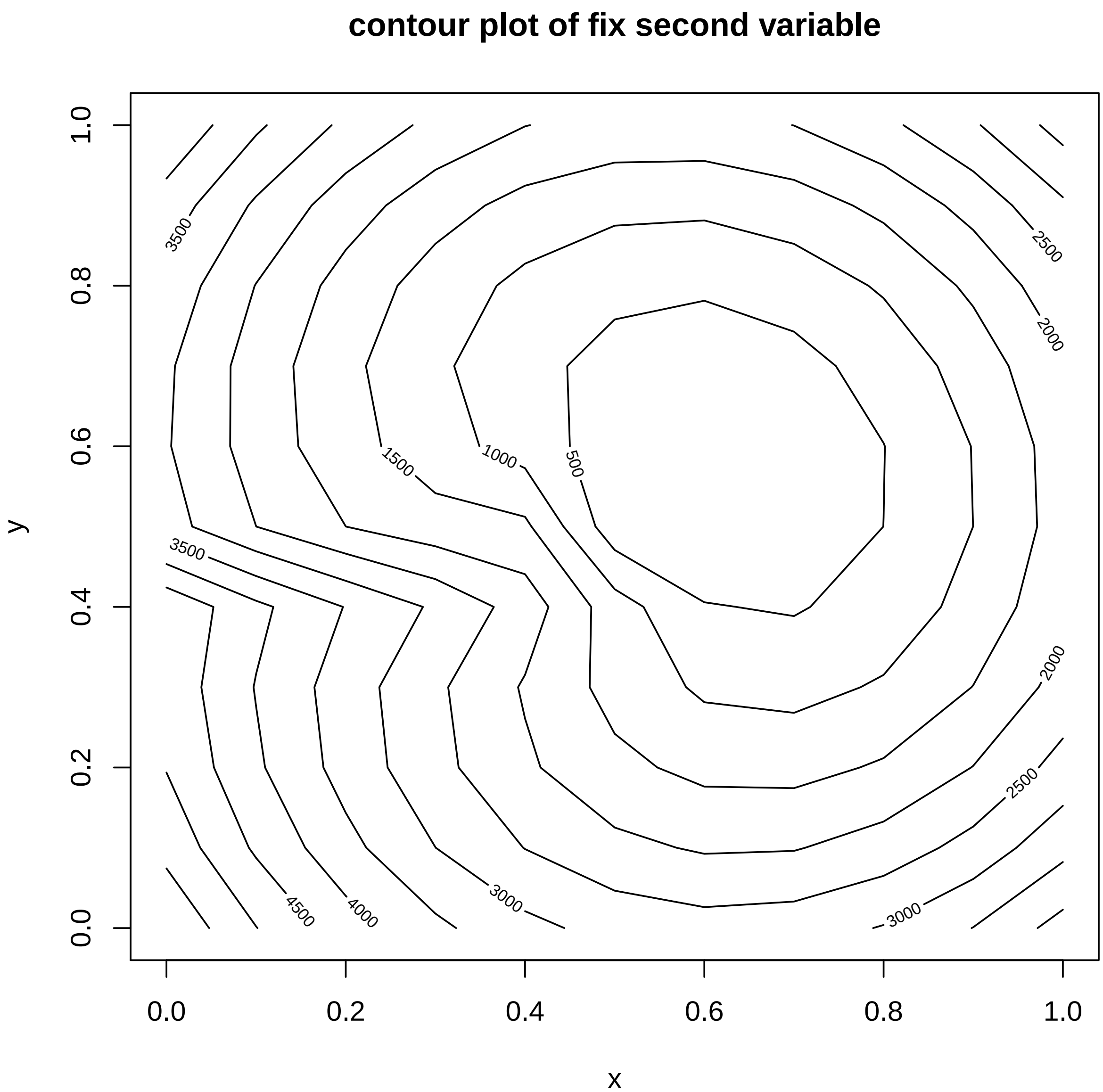

```r
# making a contour plot
contour(matrix_2,  xlab ="x", ylab = "z", main ="contour plot of fix second variable")
```




```r
# fixed the third variable as constant
Fixed_3 <- function(x,y){
  return(f(c(x,y,1)))
}
#to make the Fixed_1 able to calculate, we need to use vecterize
vectorized <- Vectorize(Fixed_3 ,vectorize.args = c("x","y"))
matrix_3<- outer(-5:5, -5:5, vectorized)

#plots of a surface over the y-z plane
persp(matrix_3, phi = 45, zlab = "fixedvalue", xlab ="y", ylab = "z")
```




```r
# making a contour plot
contour(matrix_3,  xlab ="x", ylab = "y", main ="contour plot of fix second variable")
```



From above plot we can find that if we fix one variable to 1, the function tend to increases as the rest two variables' absolute values increase. However, near the center of the two unfixed variable, the function are usually to be small.

```
#minimum by optim function.
optim(c(1,1,1), f)$par

## [1]  0.9999779414 -0.0001349269 -0.0001927127

optim(c(0,1,0), f)$par

## [1]  0.9998882225 -0.0002040018 -0.0001412321

optim(c(0,0,1), f)$par

## [1] 0.999711245 0.001178555 0.001921611
```

```r
optim(c(0,0,0), f)$par
```

```
## [1] 0.999978292 0.002730698 0.004284640
```

```r
optim(c(-1,-1,-1), f)$par
```

```
## [1] 0.9997731731 0.0005162083 0.0009457322
```

```r
optim(c(-5,-5,-5), f)$par
```

```
## [1]  1.0000614299 -0.0004356825 -0.0009109466
```

```r
optim(c(0.6,0.6,0.6), f)$par
```

```
## [1]  9.999539e-01 -3.039159e-05 -2.508388e-05
```

```r
## problem 3

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
```

```r
set.seed(0)
initial <- runif(3,-50, 50)
optim(par = initial, fn = f, control = list(trace= TRUE))
```

```
##   Nelder-Mead direct search function minimizer
## function value for initial parameters = 217653.451008
##   Scaled convergence tolerance is 0.00324329
## Stepsize computed as 3.966972
## BUILD              4 250164.956010 201295.454430
## EXTENSION          6 217653.451008 137712.813236
## LO-REDUCTION       8 209669.914815 137712.813236
## EXTENSION         10 201295.454430 86792.293623
## EXTENSION         12 148081.612739 22489.608231
## LO-REDUCTION      14 137712.813236 22489.608231
## REFLECTION        16 86792.293623 4471.532463
## REFLECTION        18 24789.301328 4188.613708
## REFLECTION        20 22489.608231 3725.104639
## HI-REDUCTION      22 4471.532463 2728.136602
## HI-REDUCTION      24 4188.613708 1351.920889
```

```
## LO-REDUCTION      26 3725.104639 358.565738
## HI-REDUCTION      28 2728.136602 358.565738
## HI-REDUCTION      30 1351.920889 358.565738
## LO-REDUCTION      32 1058.050465 285.049719
## HI-REDUCTION      34 470.535467 46.128439
## REFLECTION        36 358.565738 40.213717
## HI-REDUCTION      38 285.049719 40.213717
## HI-REDUCTION      40 92.636019 40.213717
## HI-REDUCTION      42 66.190678 24.409538
## HI-REDUCTION      44 46.128439 7.939230
## HI-REDUCTION      46 40.213717 7.939230
## HI-REDUCTION      48 24.409538 7.731708
## HI-REDUCTION      50 12.117106 6.777890
## HI-REDUCTION      52 7.939230 2.855432
## HI-REDUCTION      54 7.731708 2.048930
## HI-REDUCTION      56 6.777890 1.519195
## LO-REDUCTION      58 2.855432 1.519195
## REFLECTION        60 2.048930 1.318377
## HI-REDUCTION      62 1.834436 0.666956
## HI-REDUCTION      64 1.519195 0.666956
## HI-REDUCTION      66 1.318377 0.666956
## HI-REDUCTION      68 0.850947 0.666956
## HI-REDUCTION      70 0.736255 0.602566
## HI-REDUCTION      72 0.684598 0.579415
## HI-REDUCTION      74 0.666956 0.571048
## HI-REDUCTION      76 0.602566 0.545298
## HI-REDUCTION      78 0.579415 0.545298
## LO-REDUCTION      80 0.571048 0.543037
## REFLECTION        82 0.553831 0.527255
## HI-REDUCTION      84 0.545298 0.527255
## REFLECTION        86 0.543037 0.524988
## HI-REDUCTION      88 0.530530 0.524988
## EXTENSION         90 0.527255 0.509661
## LO-REDUCTION      92 0.526737 0.509661
## REFLECTION        94 0.524988 0.506376
## REFLECTION        96 0.516290 0.499993
## LO-REDUCTION      98 0.509661 0.499993
## EXTENSION        100 0.506376 0.489413
## LO-REDUCTION     102 0.503390 0.489413
## EXTENSION        104 0.499993 0.472072
## LO-REDUCTION     106 0.493135 0.472072
## LO-REDUCTION     108 0.489413 0.472072
## EXTENSION        110 0.477277 0.445537
## LO-REDUCTION     112 0.474771 0.445537
## LO-REDUCTION     114 0.472072 0.445537
## EXTENSION        116 0.450120 0.403716
## LO-REDUCTION     118 0.445690 0.403716
## LO-REDUCTION     120 0.445537 0.403716
## REFLECTION       122 0.419721 0.395192
## EXTENSION        124 0.408460 0.365805
## LO-REDUCTION     126 0.403716 0.365805
## LO-REDUCTION     128 0.395192 0.365805
## EXTENSION        130 0.372157 0.316378
```

```
## LO-REDUCTION     132 0.368095 0.316378
## LO-REDUCTION     134 0.365805 0.316378
## EXTENSION        136 0.327440 0.233956
## LO-REDUCTION     138 0.317164 0.233956
## LO-REDUCTION     140 0.316378 0.233956
## EXTENSION        142 0.262181 0.164547
## EXTENSION        144 0.237841 0.144745
## EXTENSION        146 0.233956 0.101887
## HI-REDUCTION     148 0.164547 0.101887
## EXTENSION        150 0.149986 0.077142
## HI-REDUCTION     152 0.144745 0.077142
## EXTENSION        154 0.105432 0.029256
## HI-REDUCTION     156 0.101887 0.029256
## LO-REDUCTION     158 0.077142 0.029256
## EXTENSION        160 0.067772 0.001110
## LO-REDUCTION     162 0.039595 0.001110
## LO-REDUCTION     164 0.029256 0.001110
## LO-REDUCTION     166 0.016071 0.001110
## LO-REDUCTION     168 0.010525 0.001110
## Exiting from Nelder Mead minimizer
##     170 function evaluations used
## $par
## [1]  0.998334136 -0.004430611 -0.008296619
##
## $value
## [1] 0.0004951965
##
## $counts
## function gradient
##      170       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optim(par = initial, fn = f, control = list(trace= TRUE), method = 'BFGS' )
```

```
## initial  value 217653.451008
## iter  10 value 816.973113
## iter  20 value 6.060495
## iter  30 value 0.248793
## iter  40 value 0.000000
## final  value 0.000000
## converged
## $par
## [1] 1.000000e+00 2.849319e-13 4.334081e-13
##
## $value
## [1] 3.504477e-25
##
## $counts
## function gradient
```

```
##      113       42
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
nlm(f, p= initial)
```

```
## $minimum
## [1] 1.700918e-08
##
## $estimate
## [1]  0.9999994970 -0.0000822318 -0.0001300803
##
## $gradient
## [1]  4.612353e-08  3.302012e-08 -2.128825e-08
##
## $code
## [1] 1
##
## $iterations
## [1] 28
```

Problem 3

b)

In order to find a reasonable starting value, we can ingore the censored data and run the regression of the uncensored values. We used the estimated parameter we got from the regression as our starting values.

c)

```
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
```

```r
summary(mod)$sigma

## [1] 2.305117

ycomplete1 <- yComplete


myfunction <- function(x, y, p, criteria){
  #censore the data based on the percentile we want to use
  ycomplete1[yComplete > quantile(yComplete, p)] <- unname(quantile(yComplete, p))
  #calculate the initial parameter value
  initial <- lm(ycomplete1~x)
  beta1 <- unname(initial$coefficients[2])
  beta0 <- unname(initial$coefficients[1])
  sigma <- summary(initial)$sigma
  n <- length(yComplete)

  # Threshold
  tau <- unname(quantile(yComplete, p))
  difference = 10
  i = 0
  while( difference > criteria){
    #print(c(beta1,beta0,sigma, i))
    # Calculate expected value and variance of truncated normal distribution
    mu <- beta0 + beta1*x
    tau2 <- (tau - mu) / sigma
    rho <- dnorm(tau2) / (1 - pnorm(tau2))
    Expectation <- mu + sigma*rho
    Var <- sigma^2 *(1 + tau2*rho - (rho)^2)

    #step of maximization
    ycomplete1[yComplete > tau]<- Expectation[yComplete > tau]
    #Do the regression again after the modification to calculate the coefficient beta
    Update <- lm(ycomplete1 ~ x)
    beta1_new <- unname(Update$coefficients[2])
    beta0_new <- unname(Update$coefficients[1])
    sigma_new <- sqrt(sum(Var[yComplete > tau])/n + sum((ycomplete1 - beta0_new - beta1_new * x)^2)/n)

    #calculate the difference of the parameter to check if the parameter converge
    diff_beta1 <- abs(beta1 - beta1_new)
    diff_beta0 <- abs(beta0 - beta0_new)
    diff_sigma <- abs(sigma - sigma_new)

    difference <- (diff_beta1 + diff_beta0 + diff_sigma)

    #assign the new parameter to the parameter will be used in next iteration
    beta1 <- beta1_new
    beta0 <- beta0_new
    sigma <- sigma_new

    i = i + 1
```

```r
  }
  return(c(beta0_new, beta1_new, sigma_new, i-1))
}

result1 <- myfunction(x,y,0.2,0.00000001)
cat("Beta0: ",result1[1],'Beta1: ', result1[2], 'sigma: ', result1[3], 'iteration: ', result1[4])

## Beta0:  0.3126394 Beta1:  2.87922 sigma:  1.960093 iteration:  208

result2 <- myfunction(x,y,0.8,0.00000001)
cat("Beta0: ",result2[1],'Beta1: ', result2[2], 'sigma: ', result2[3], 'iteration: ', result2[4])

## Beta0:  0.4566128 Beta1:  2.824108 sigma:  2.149157 iteration:  16
```

The complete data's beta0, beta1, and sigma are 0.5607422, 2.7650812, and 2.305117 respectively. when a modest 20 percent proportion of exceedances expected, beta0, beta1, and sigma are 0.3126394, 2.87922, and 1.960093 respectively. I did 208 times iteration. when a high 80 percent proportion of exceedances expected, beta0, beta1, and sigma are 0.4566128, 2.824108,2.149157 respectively. I only did 16 times iteration. I choose the difference criteria as 0.0000001, becuase I think it small enough.

d)

```r
myfunction_logL  <- function(x, y, p=0.8){
  ycomplete1[yComplete > quantile(yComplete, p)] <- unname(quantile(yComplete, p))
  #calculate the initial parameter value
  initial <- lm(ycomplete1~x)
  beta1 <- unname(initial$coefficients[2])
  beta0 <- unname(initial$coefficients[1])
  sigma <- summary(initial)$sigma
  n <- length(x)
  tau <- unname(quantile(yComplete, p))
  mu <- rep(beta0, n) + beta1*x
  y <- (yComplete[1:(n*(1-p))] - mu[1:(n*(1-p))]) / sqrt(sigma)
  return(n*(1-p)*log(sqrt(sigma)) + sum(y^2)/2 - sum(log(1 - pnorm((tau - mu[((n*(1-p))+1):n]) / sqrt(s
}
  ycomplete1[yComplete > quantile(yComplete, 0.8)] <- unname(quantile(yComplete, 0.8))
  #calculate the initial parameter value
  initial <- lm(ycomplete1~x)
par <- c(unname(initial$coefficients[2]), unname(initial$coefficients[1]),summary(initial)$sigma)
optim(par, fn = myfunction_logL, method = "BFGS", x = x,y =ycomplete1)

## Error in quantile.default(yComplete, p):  'probs' outside [0,1]
```