ps8

Kehsin Su Esther 3033114294

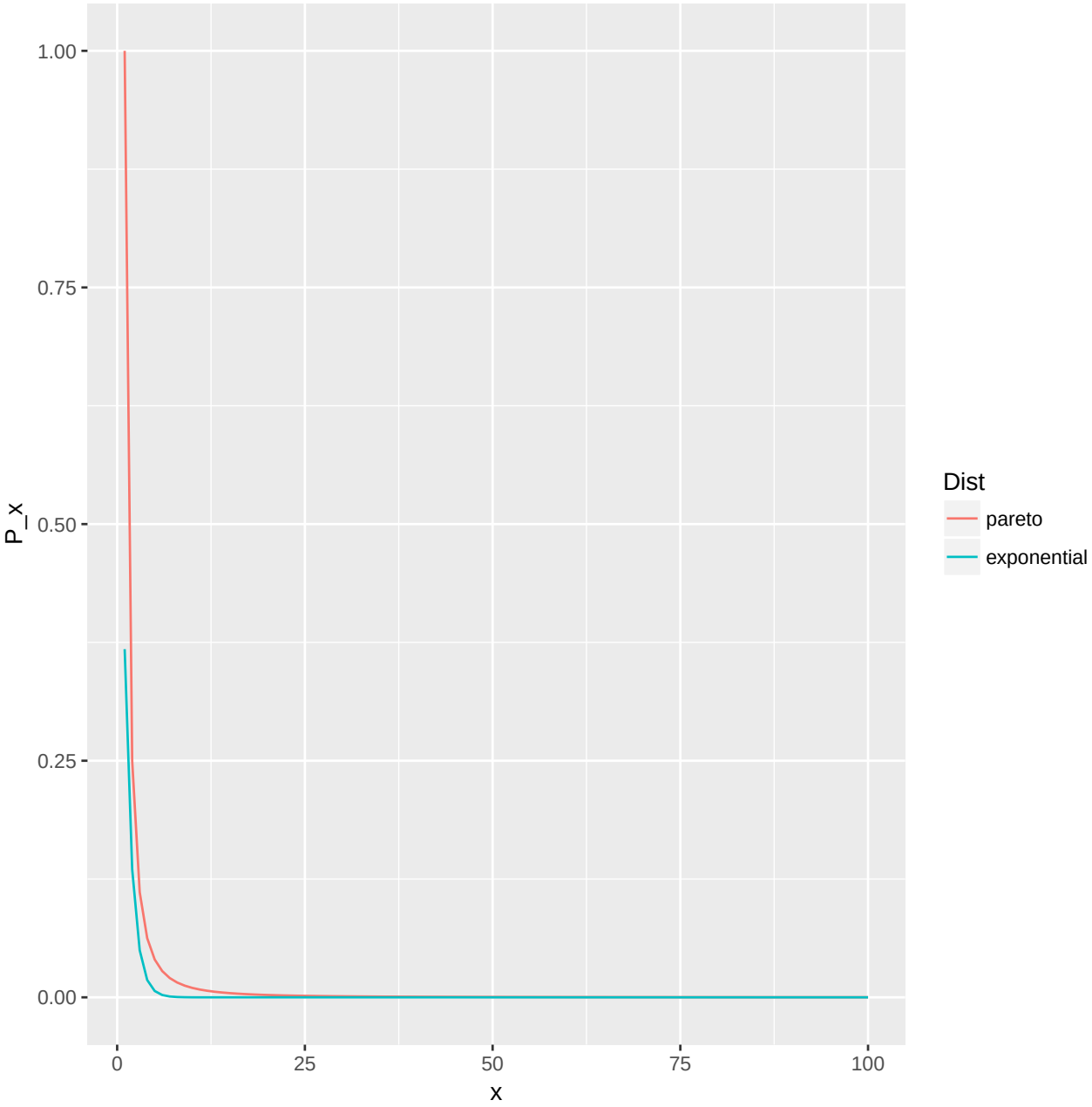
November 30, 2017

1 Q1

1.1 (a)

As the pdf of exponential distribution has exponential term and pareto is polynomial terms, so when x goes into infinite, exponential distribution will goes to zero faster. Thus, Pareto distribution has heavior tail. We can use it to do important sample for exponential distribution, which has lighter tail. As showing is the following graph, y1 has heavior tail.

```
require(sads)

## Loading required package:  sads
## Loading required package:  bbmle
## Loading required package:  stats4

require(ggplot2)

## Loading required package:  ggplot2

require(reshape2)

## Loading required package:  reshape2

x <- 1:100
pareto <- sads::dpareto(x, shape=1, scale = min(x), log = FALSE)
exponential <- dexp(x, rate = 1, log = FALSE)
df <- melt(data = data.frame(x,pareto,exponential), id.vars = "x")
colnames(df) <- c("x","Dist","P_x")
ggplot(data = df, aes(x = x, y = P_x, colour = Dist)) + geom_line()
```

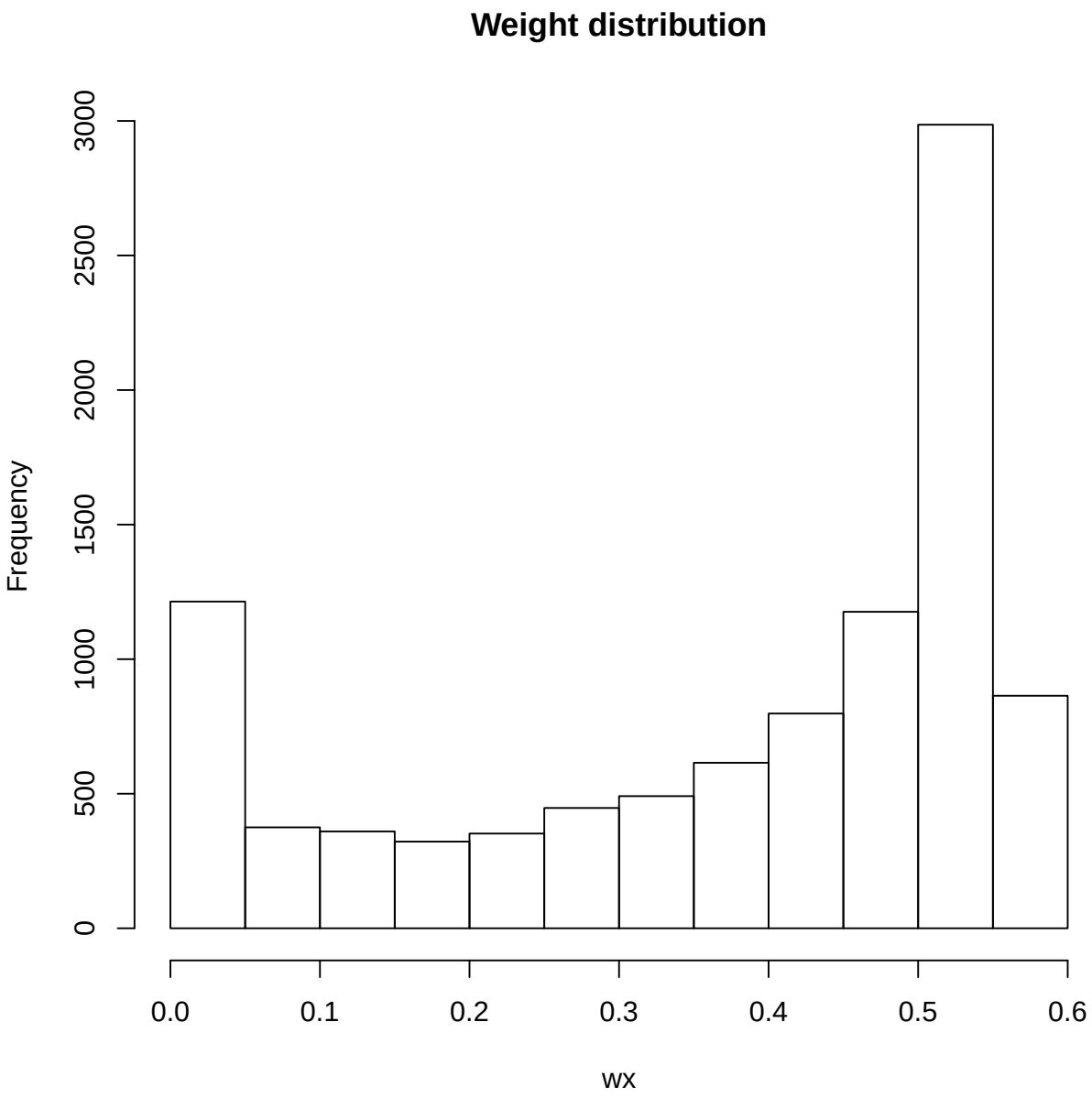


1.2 (b)

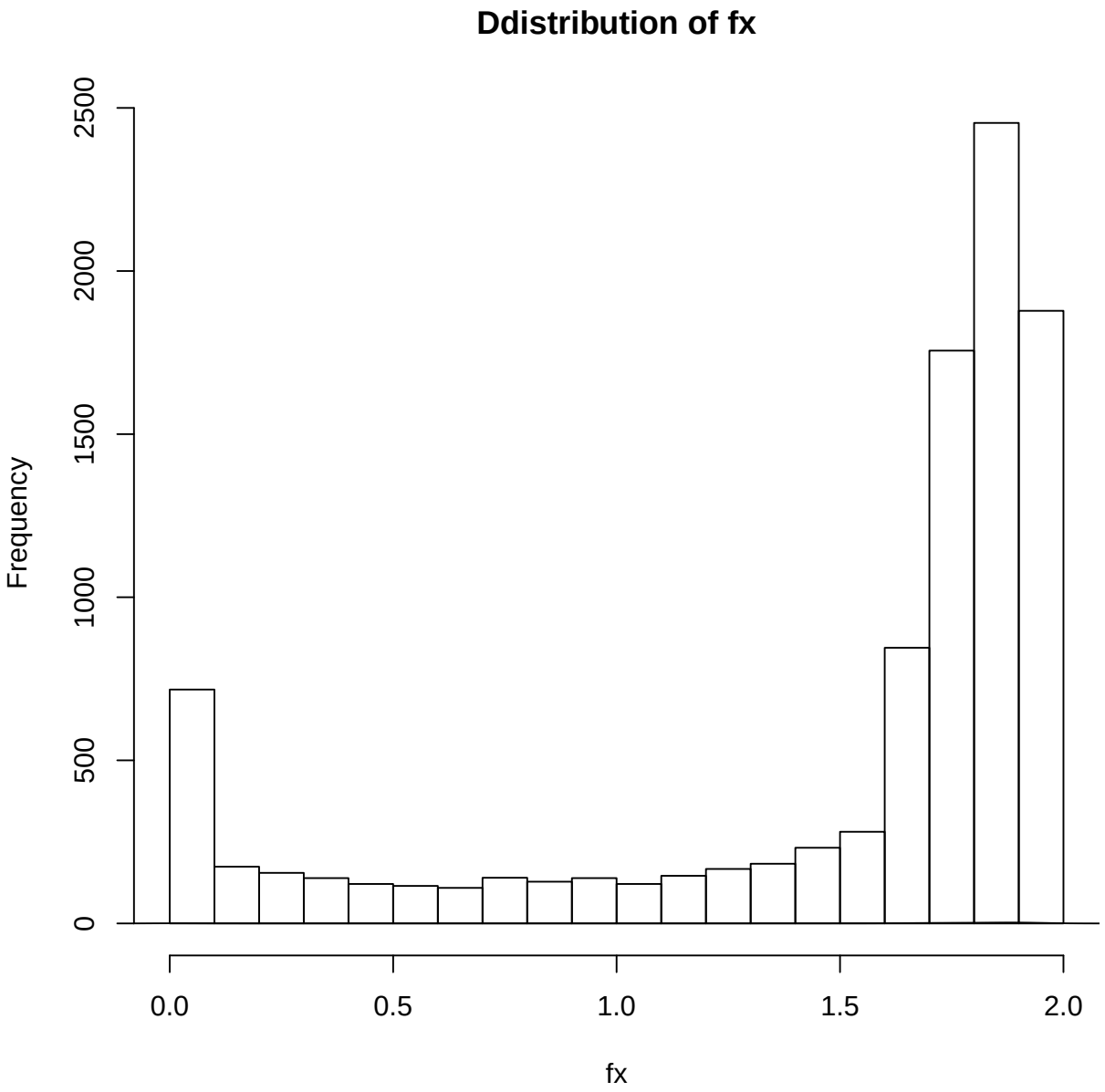
As showing in the following graph, the weight is greater than on the right hand side, but generally, almost all the part has been sampled. Thus, it is fine to use pareto to sampling exponential distribution.i.e. it is fine to use a distribution with heavior tail to sample a distribution with lighter tail, but it won't hold when we switch two distributions to sample.

```
library(tolerance)
m <- 10000
a <- 2
b <- 3
theta <- 1
shift <- 2
#fx <- r2exp(m, rate = 1, shift = 2)
x <- rpareto(m, shape=a, scale = b)
```

```r
hx <- x
wx <- d2exp(x, rate = theta, shift = shift)/dpareto(x, shape=a, scale = b) #weight
hist(wx, main='Weight distribution')
```




```r
fx <- hx * wx
hist(fx,main='Ddistribution of fx')
lines(density(fx))
```




```
(ex_hat <- mean(fx))
```

```
## [1] 1.482589
```

```
(vx_hat <- var(fx))
```

```
## [1] 0.3639374
```

```
(ex_hat2 <- ex_hat^2+vx_hat)
```

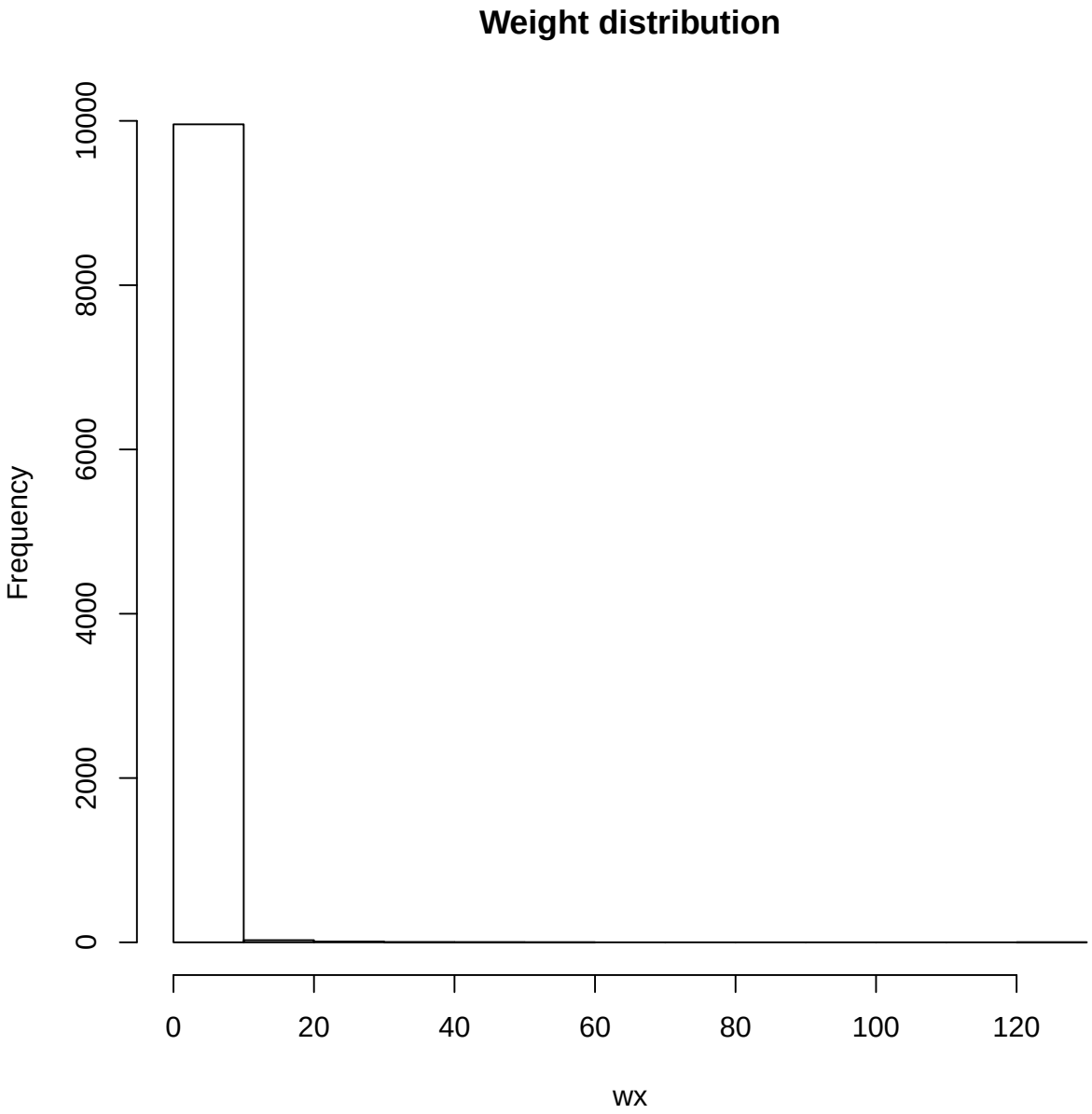
```
## [1] 2.562008
```

```
ex <- theta + shift
vx <- theta
mse <- (ex_hat-ex)^2
```

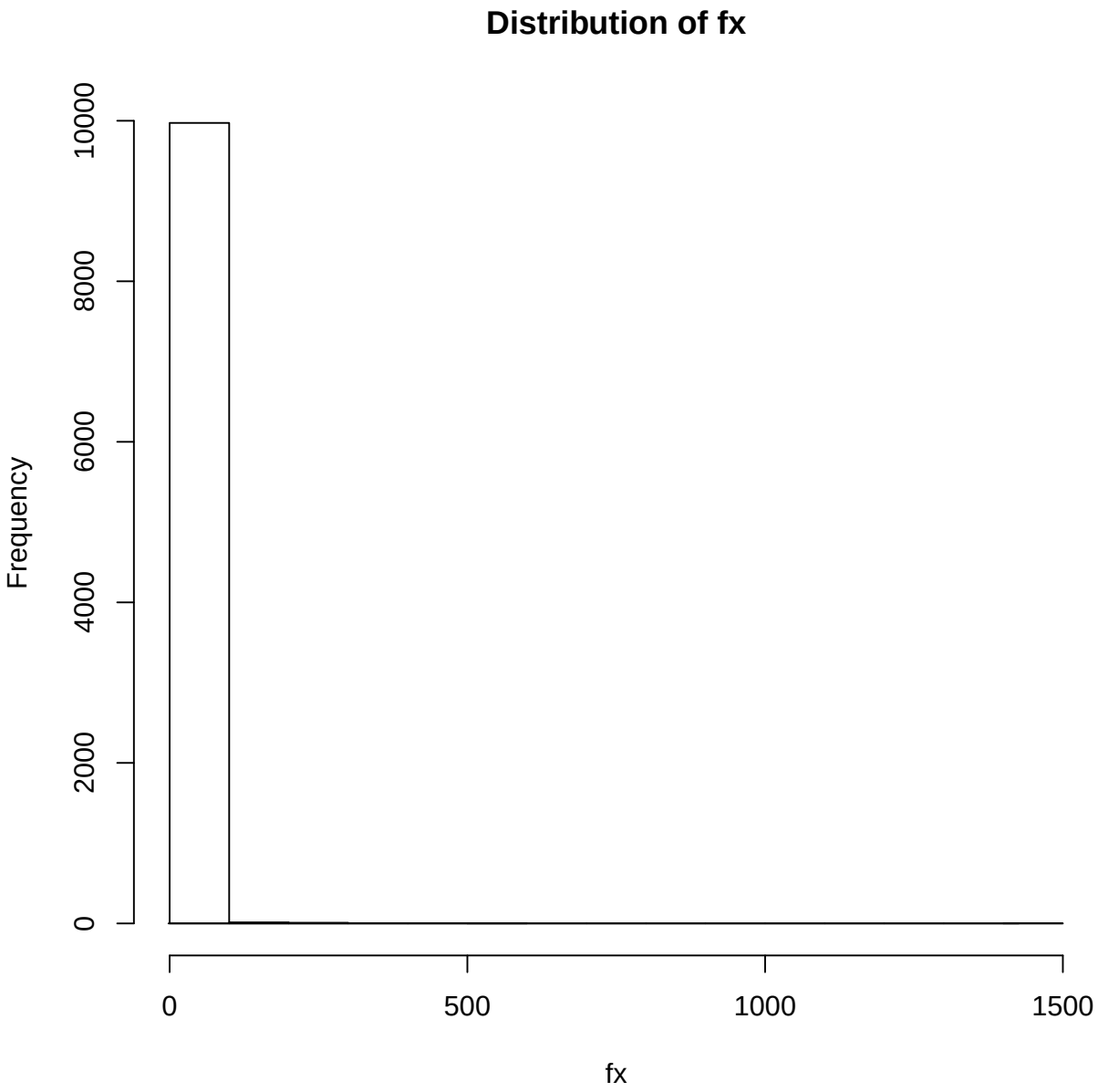
1.3 (c)

As showing in the following graph, as the one with heavior divide the one with ligter tail will make the value go to infinite, so as the histgram, it center mostly on the left sight which close to zero.

```
library(tolerance)
m <- 10000
x <- r2exp(m, rate = theta, shift = shift)
#x <- rpareto(m, shape= a, scale = b)
hx <- x
wx <- dpareto(x, shape=a, scale = b)/d2exp(x, rate = theta, shift = shift) #weight
hist(wx, main='Weight distribution')
```




```
fx <- hx * wx
hist(fx,main='Distribution of fx')
lines(density(fx))
```




```
(ex_hat <- mean(fx))
## [1] 4.29801
(vx_hat <- var(fx))
## [1] 402.0106
(ex_hat2 <- ex_hat^2+vx_hat)
## [1] 420.4835
(ex <- b*a/(b-1))
## [1] 3
(vx <- b*a^2/(b-1)^2/(b-2))
```

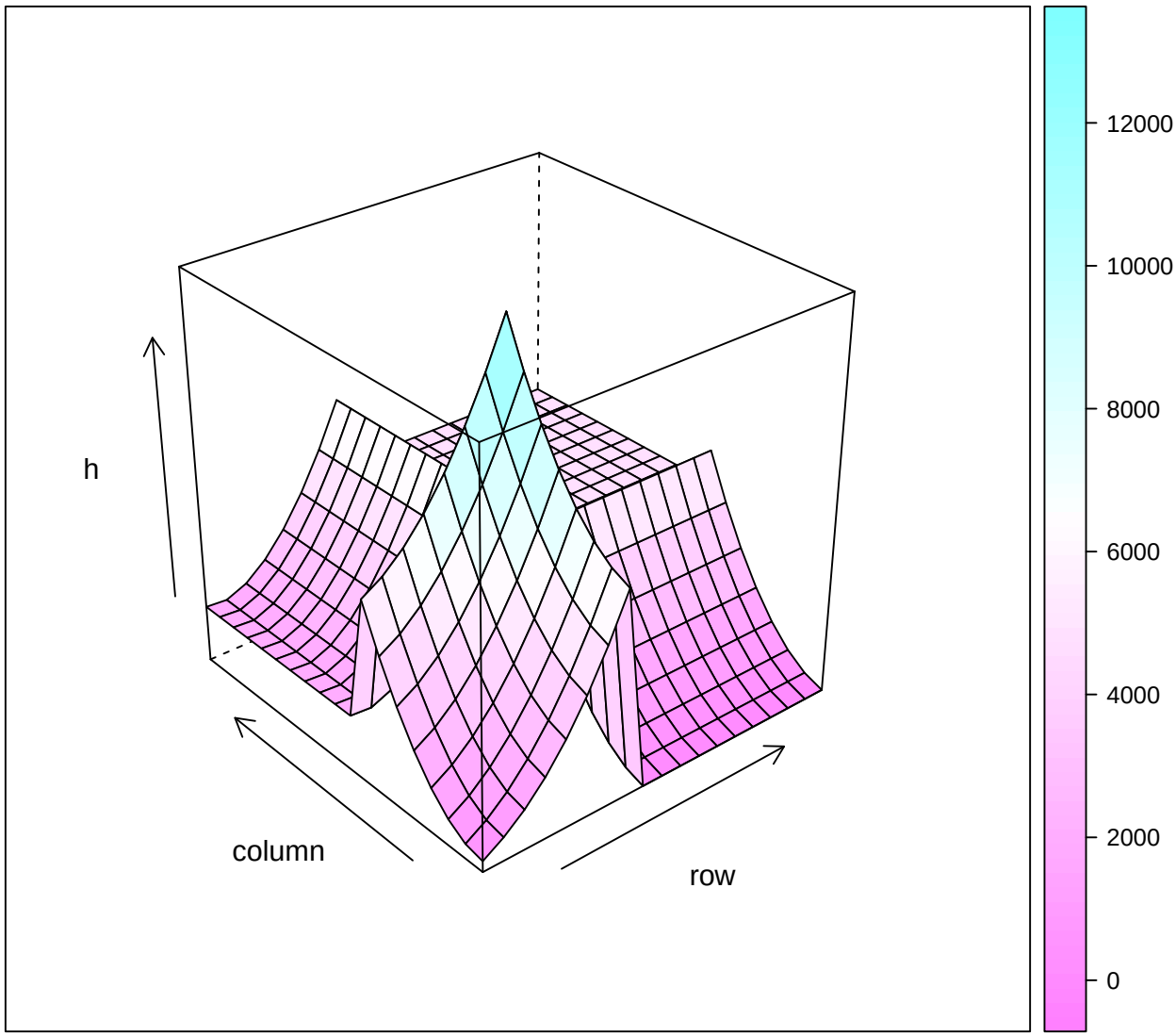
```
## [1] 3

ex2 <- ex^2+vx
(mse <- (ex_hat-ex)^2)

## [1] 1.68483
```

2 Q2

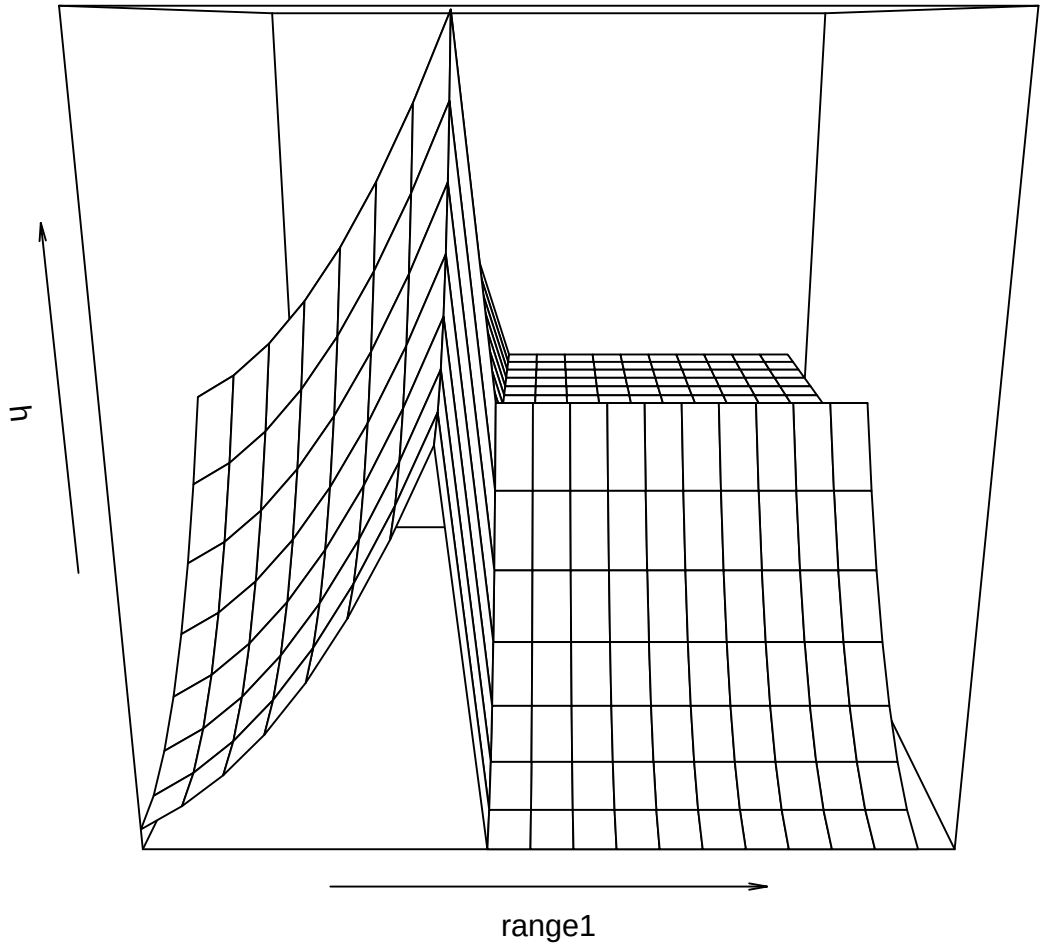
First, fix a dimension and plot some 3d plot to observe how does the curve looks like. Then, we found that it's hard to identify the lowest point. Contour make it easier to identify to lowest point. With differnet slice, we plot different picture and choose to possible point to that dimension.

```
library(rgl)
library(graphics)
library(ggplot2)
## 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)
}
## calculate the minimum

f2 <- 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)
}
my_filled_plot <- function(x) {
  range1 <- seq(-3*pi,3*pi,1)
  n1 <- length(range1)
  h <- matrix(NA,n1,n1)
  d3 <- x
  for (i in range1){
    for(j in range1){
      h[i,j] <- f2(c(i,j,d3))
    }

  }
  filled.contour(range1,range1,h, color = terrain.colors)

}

range1 <- seq(-3*pi,3*pi,1)
n1 <- length(range1)
h <- matrix(NA,n1,n1)
d3 <- pi
for (i in range1){
  for(j in range1){
    h[i,j] <- f2(c(i,j,d3))
```

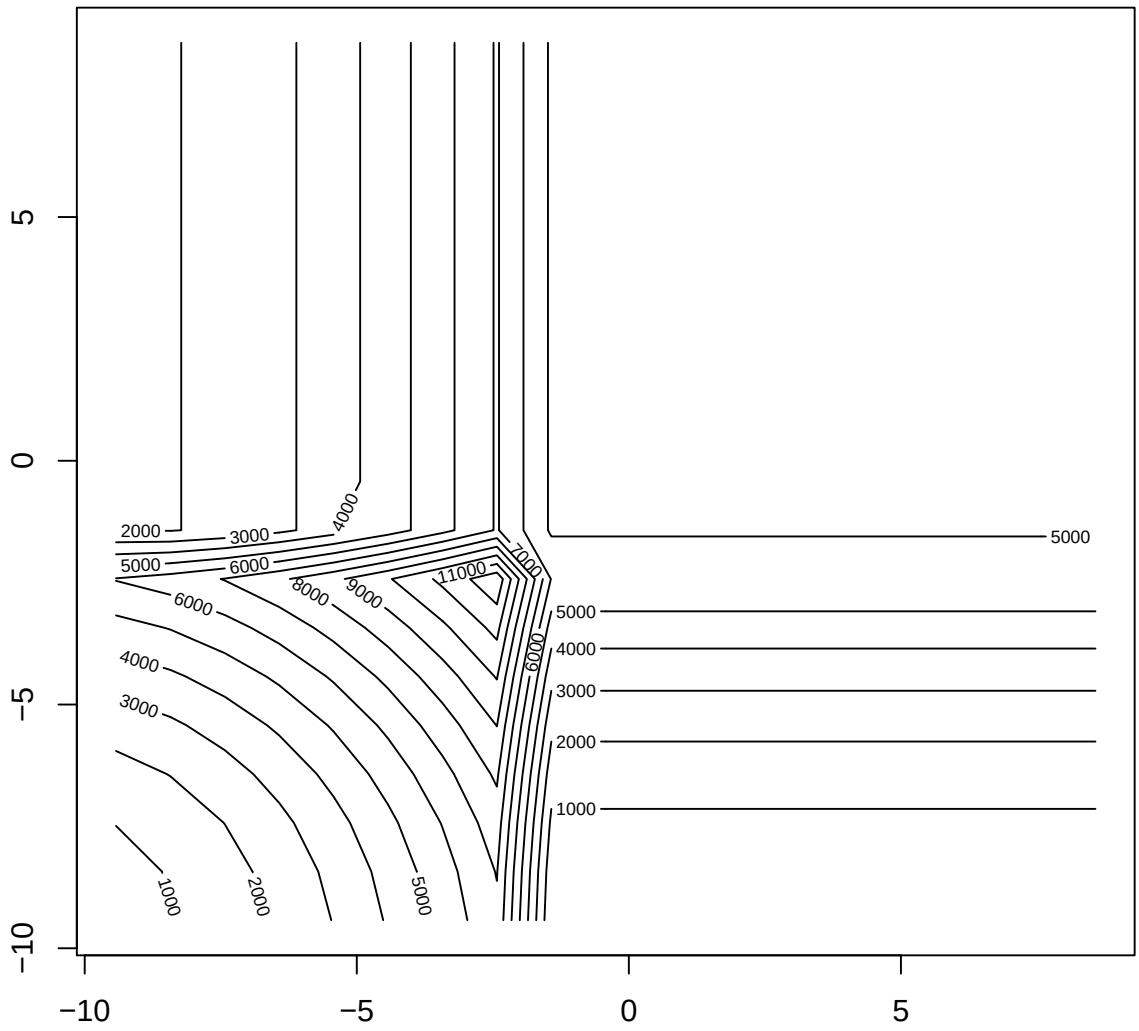
```
  }
}
require(lattice)

## Loading required package:  lattice

#wireframe(h, drape=T, col.regions=rainbow(10000000))
wireframe(h, drape=T)
```



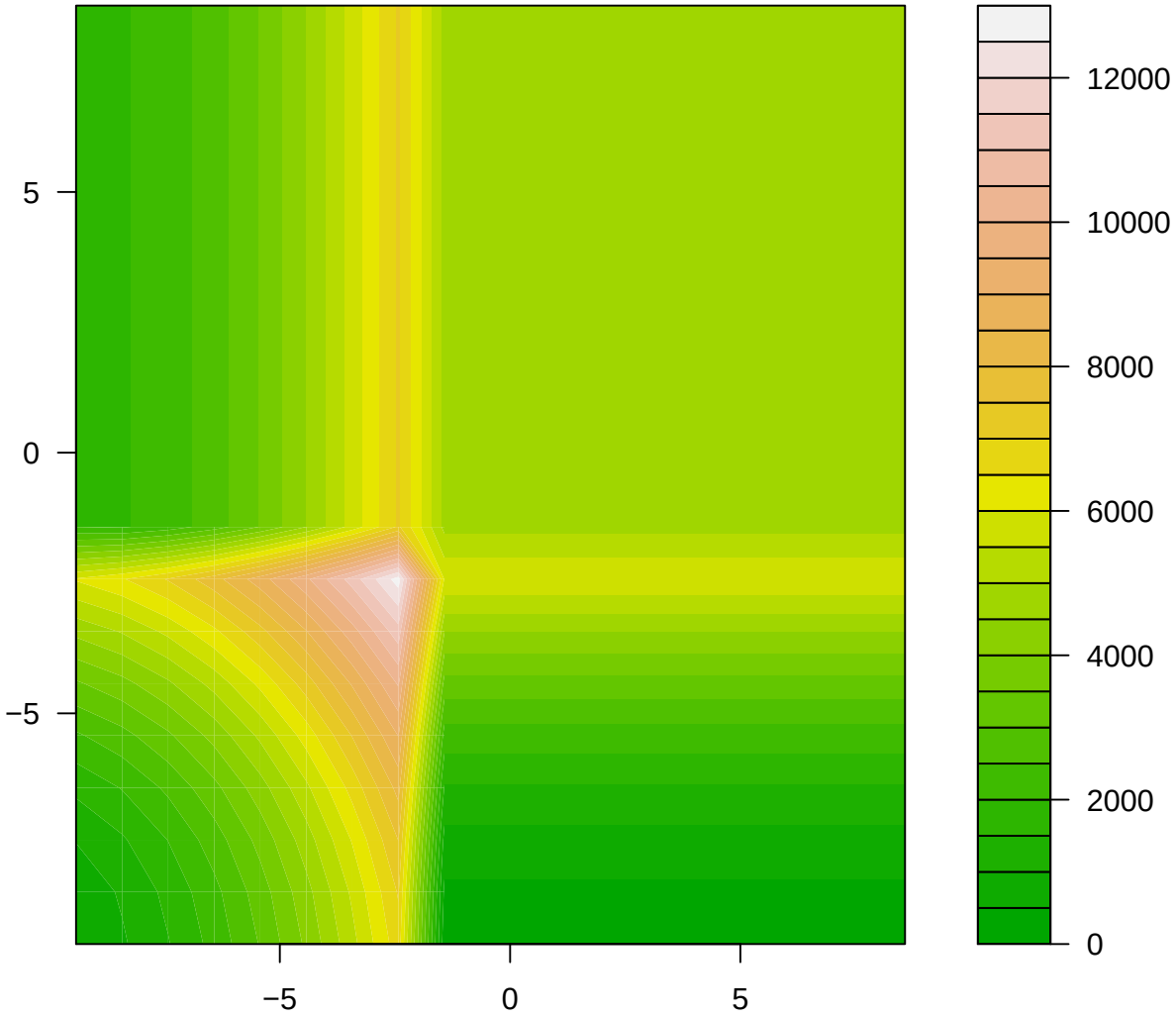

```
persp(range1,range1,h,xlim = range(-3*pi,3*pi), ylim = range(-3*pi,3*pi))
```




```
contour(range1,range1,h)
```




```
my_filled_plot(pi)
```




```
# require(rgl)
# open3d()
# rgl.surface(range1,range1,h)

require(plot3D)

## Loading required package:  plot3D

#persp3D(z = h, theta = 120)

library(graphics)
library(ggplot2)
myplot1 <- function(x1,x2,x3){
  x <- list(x1 ,x2, x3)
  z <- f(x)
  df <- data.frame(x1,x2,z)
```

```
  plot3d(df)
  }

myplot <- function(x1,x2,x3){
  x <- list(x1 ,x2, x3)
  z <- f(x)
  height <-cut(z,20)
  df <- data.frame(x1,x2,z)
  (ggplot(data=df, aes(x=x1, y=x2, z=z))
  + geom_point(aes(colour=height))
  + geom_density2d()

  )
}
#test fixed x3 in four different value: 0,pi/2, 3pi/2, 2pi
#since we need to calculate tan of x1,x2, so it's better to set x3 within the range from -2pi to 2pi to
x1 <- seq(-2*pi, 2*pi, length= 100)
x2 <- seq(-2*pi, 2*pi, length= 100)
x3 <- rep(0,100)
#myplot(x1,x2,x3)

x3 <- rep(pi,100)
#myplot(x1,x2,x3)

x3 <- rep(2*pi,100)
#myplot(x1,x2,x3)

x3 <- rep(-pi,100)
#myplot(x1,x2,x3)

x3 <- rep(-2*pi,100)
#myplot(x1,x2,x3)

# par(mfrow=c(2,2))



#When fixed x3 equal to zero, the color chenge at about point (0,0) and point (2.5,2.5), so we set thes
p1 <- c(0,0,0)
optim(p1,f)$value

## [1] 1.876851e-05

nlm(f,p1)$minimum

## [1] 100

p2 <- c(2.5,2.5,0)
optim(p2,f)$value

## [1] 3.217961e-06

nlm(f,p2)$minimum #better

## [1] 3.348995e-19

#When fixed x3 equal to pi, the color chenge at about point (0,0) and point (2,2), so we set these two
p3 <- c(0,0,pi)
optim(p3,f)$value
```

```
## [1] 1.199054e-05

nlm(f,p3)$minimum

## [1] 149.8989

p4 <- c(2,2,pi)
optim(p4,f)$value

## [1] 9.900303e-07

nlm(f,p4)$minimum #better

## [1] 1.294807e-18

#When fixed x3 equal to pi*2, the color chenge at about point (0,0) and point (2,2), so we set these tw
p5 <- c(0,0,2*pi)
optim(p5,f)$value

## [1] 1.253867e-05

nlm(f,p5)$minimum

## [1] 1569.631

p6 <- c(2,2,2*pi)
optim(p6,f)$value

## [1] 1.264891e-05

nlm(f,p6)$minimum #better

## [1] 2.129973e-18

#When fixed x3 equal to -pi, the color chenge at about point (0,0) and point (-2,6), so we set these tw
p7 <- c(-2,-2,-pi)
optim(p7,f)$value

## [1] 4.910886e-06

nlm(f,p7)$minimum #better

## [1] 1.700644e-08

p8 <- c(0,0,-pi)
optim(p8,f)$value

## [1] 6.625893e-06

nlm(f,p8)$minimum

## [1] 150.5356

#When fixed x3 equal to -2*pi, the color chenge at about point (0,0) and point (-1.5,-1.5), so we set t
p9 <- c(-1.5,-1.5,-pi*2)
optim(p9,f)$value

## [1] 5.694121e-05

nlm(f,p9)$minimum #better

## [1] 1.700871e-08
```

```r
p10 <- c(0,0,-pi*2)
optim(p10,f)$value

## [1] 1.179725e-05

nlm(f,p10)$minimum

## [1] 1569.67

#As showing above, set p6 as beginning point will generates the mimimum value, so we choose it with nlm
nlm(f,p2)

## $minimum
## [1] 3.348995e-19
##
## $estimate
## [1] 1.000000e+00 1.820975e-10 2.380022e-10
##
## $gradient
## [1] -1.977529e-09  1.649318e-08 -9.886966e-09
##
## $code
## [1] 1
##
## $iterations
## [1] 22
```

3 Q3

3.1 (a)

Details append at last page. The first step is to condidition on the coefficients and calcuate the expectation of log likelihood funciton. Second, we can take log and solve the equation to get the recursive solution of coefficients. The key concept is that the coefficent can be evalued similar like general regression coefficent, but the cencor part is estimated by specific expectation value to start em algorithm, and variance also has a special term for the cencor part.

3.2 (b)

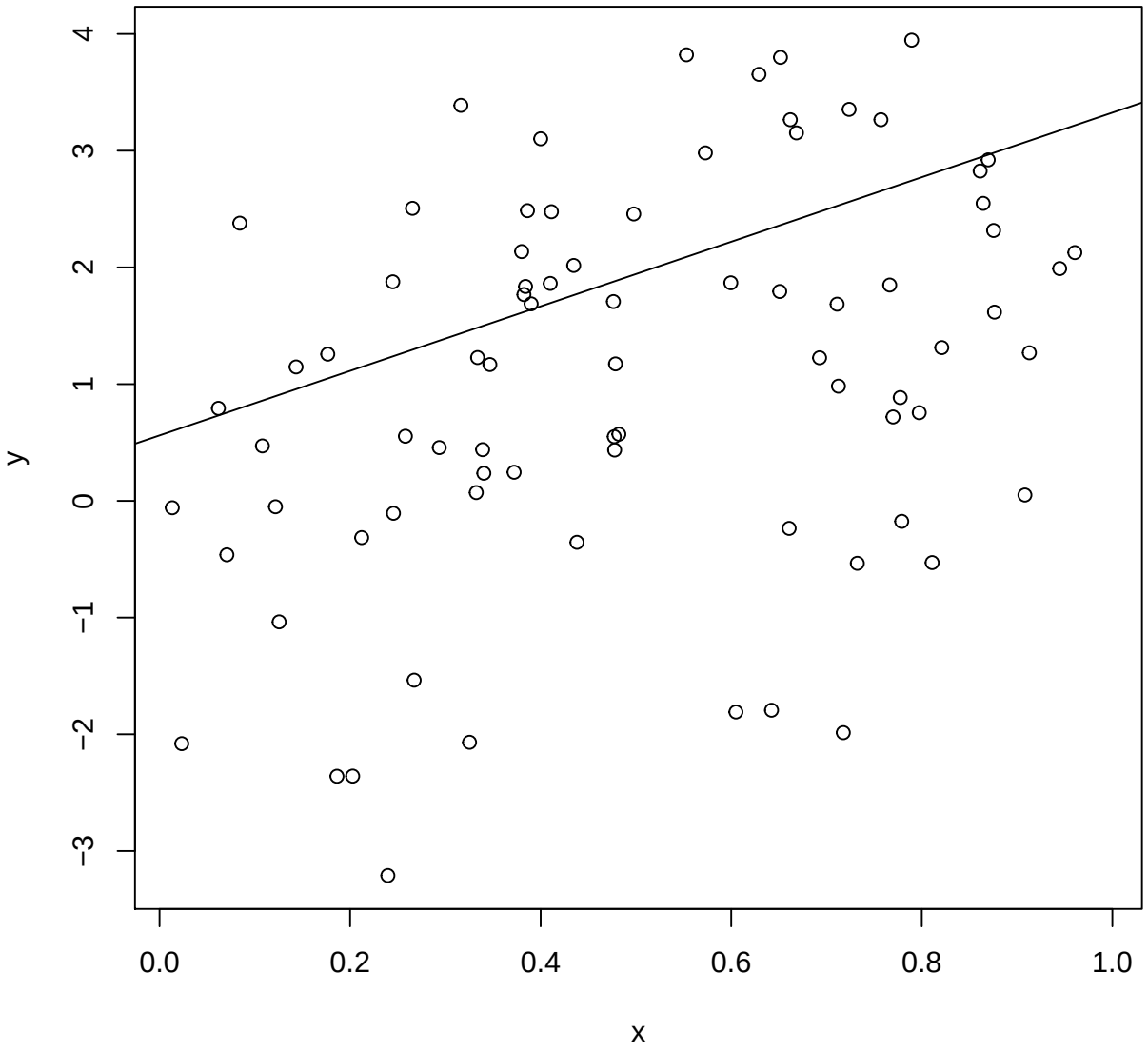
The start point can be set by uncencoring data. We can use summary of "lm" function to get beta0, beta1 and sigma as an initial point. Then use such coefficeints to estimate the cencor data and keep updating the coefficients until to the coefficients very close to the coefficients we obtain in the previous step. Then we can get the final solution.

3.3 (c)

```r
# generate the testing data for the results and assumption we make on (a) and (b)
set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

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

```r
## parameters chose such that signal in data is moderately strong
## estimate divided by std error is ~ 3
mod <- lm(yComplete ~ x)
beta0 <- summary(mod)$coef[1]
beta1 <- summary(mod)$coef[2]
sigma2 <- summary(mod)$sigma^2
c(beta0,beta1, sigma2)

## [1] 0.5607442 2.7650812 5.3135623

#change 20% data to censor
y <- yComplete
cp <- quantile(y,0.79)[[1]]+10^-3 #use 79 quantile plus some eplison
y[y>cp]<- NA #set the censor data into NA

cen <- function(x,y,cp){
  #beta0 <- 0 #initiaization
  #beta1 <- 0 #initiaization
  #sigma <- 0 #initiaization
  cpp <- which(is.na(y)) #censor position
  diff <- 10 #threshold, setting a default to make while loop start to run
  c <- length(cpp)
  n <- length(x)
     #ignore cencoring part and fit model to get initial coefficients
    est <- lm(y ~ x, na.action=na.omit)
    gbeta0 <- summary(est)$coef[1]
    gbeta1 <- summary(est)$coef[2]
    gsigma2 <- summary(est)$sigma^2
    #declare empty list
    gmu <- vector("list",c)
    gthou <- vector("list",c)
    glo <- vector("list",c)
    v <- vector("list",c)
    while(diff!=1){ #threshold
    for (i in 1:c) {
      gmu[[i]] <- gbeta0+gbeta1*x[cpp[i]]
      gthou[[i]] <- (cp-gmu[[i]])/sqrt(gsigma2)
      glo[[i]] <- dnorm(gthou[[i]])/(1-pnorm(gthou[[i]]))
      y[[i]] <- gmu[[i]]+sqrt(gsigma2)*glo[[i]]
      v[[i]] <- gsigma2*(1+gthou[[i]]*glo[[i]]-glo[[i]]^2)
    }
      #re-calcuate coefficients again
      est <- lm(y ~ x)
      beta0 <- summary(est)$coef[1]
      beta1 <- summary(est)$coef[2]
      sigma2 <- (sum(sum(summary(est)$residual^2))+sum(unlist(v)))/n
      # setting small threshold as we already know the coefficients is small
      diff <- (abs(gbeta0-beta0)<10^-5)*(abs(gbeta1-beta1)<10^-5)*(abs(gsigma2-sigma2)<10^-5)
      #put coefficients into to the start point for next calculation
      gbeta0 <- beta0
      gbeta1 <- beta1
      gsigma2 <- sigma2
    }
    return(c(beta0,beta1,sigma2,diff))
}
```

```
result1 <- cen(x,y,cp)
plot(x,y)
abline(beta0,beta1)
```




```
rbind(c(beta0,beta1, sigma2),result1[1:3])

##           [,1]     [,2]     [,3]
## [1,] 0.5607442 2.765081 5.313562
## [2,] 0.5180308 2.917809 4.251793
```

3.4 (d)

```r
max_logcen <- function(parb){
  #put data into the function
  x <- x
  y <- y
  cp <- cp
  n <- length(x) #length of data
  cpp <- which(is.na(y)) #position of cencoring data
  ncpp <- which(!is.na(y)) #position of non-cencoring data
  c <- length(cpp) #length of cencoring data
  uncen <- vector("list", n-c)
  cen <- vector("list", c)
  #(need expontial back)
  s2 <- exp(parb[3]) # since sigma can not be negative, we take log
  #seperate the log likelihood function into three parts
  for  (i in 1:n-c){
    #non-censoring part
    uncen[i] <- -1/2/s2*(y[ncpp[i]]-parb[1]-parb[2]*x[ncpp[i]])^2

  }
  for (i in 1:c){
    #censoring part
    cen[i] <- log(1- pnorm((cp-parb[1]-parb[2]*x[cpp[i]])/sqrt(s2)))
  }

  logfun <- -(n-c)/2*log(2*pi)-(n-c)/2*log(s2)+sum(unlist(uncen))+sum(unlist(cen))
  return(-logfun)
}

start <- c(1,1,log(3)) #start point
#use BFGS method to calculate and try parscale
(result2 <- optim(par=start, fn=max_logcen, method = "BFGS",control = list(trace = TRUE, parscale = c(1

## initial  value 203.079711
## final  value 194.276775
## converged
## [1] 0.4948443 2.7554372 1.5402094

#try nlm and optim to compare the result
nlm(max_logcen,c(1,1,3))$estimate

## [1] 0.4947069 2.7556863 1.5392725

optim(c(1,1,3),max_logcen)$par

## [1] 0.4942796 2.7560622 1.5391427

rbind(result1[1:3],c(result2[1:2],exp(result2[3])))

##           [,1]     [,2]     [,3]
## [1,] 0.5180308 2.917809 4.251793
## [2,] 0.4948443 2.755437 4.665567

#As showing above, the result is quite similar, we can conclude that both methods work.
```