243 PS8

Hsieh Cheng Han

November 30, 2017

1.(b)

```r
# Using Pareto density function
library(actuar)

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

# First we estimate E(X), m = 10000
m <- 10000
scale <- 2
f <- function(x) dexp(x - scale)
shape <- 3
g <- function(x) dpareto(x - scale, shape, scale)
xsample <- vector(mode = "numeric", length = m)
x2sample <- vector(mode = "numeric", length = m)
fgsample <- vector(mode = "numeric", length = m)
# Bootstrap for m times
for(i in 1 : m){
  x <- rpareto(n = 1, shape, scale) + scale
  fgsample[i] <- f(x) / g(x)
  xsample[i] <- x * f(x) / g(x)
  x2sample[i] <- (x^2) * f(x) / g(x)
}
mean(xsample) # Should be close to 3
```

```
## [1] 3.010469
```

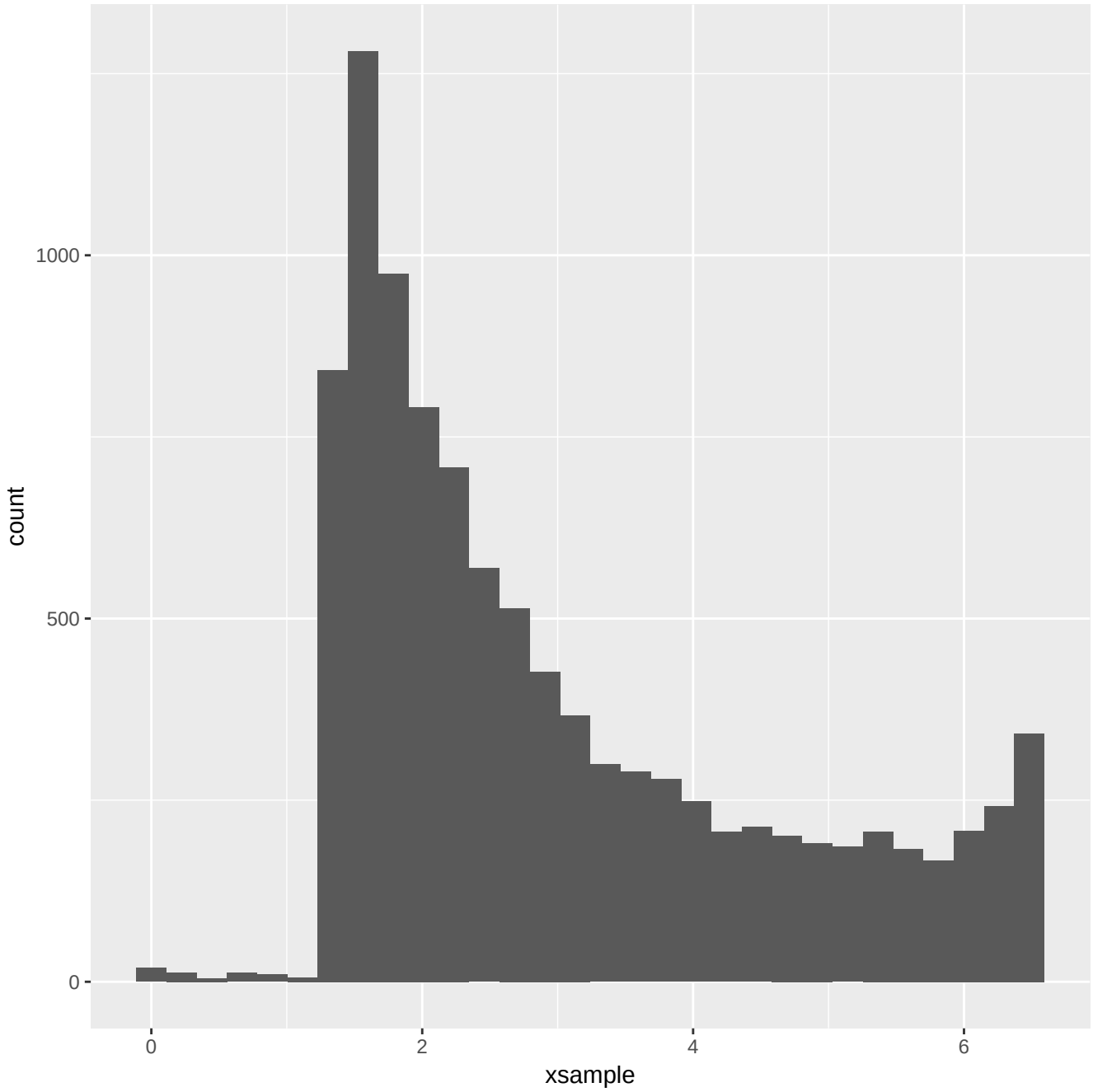
```
mean(x2sample) # Should be close to 10
```

```
## [1] 10.05364
```

For exponential distribution with parameter value = 1 and shifted by 2, E(X) = 1 + 2 = 3 and E(X^2) = Var(X) + $E(X)^2$ = 1 + 9 = 10. The bootstrap method gives a good approximation.

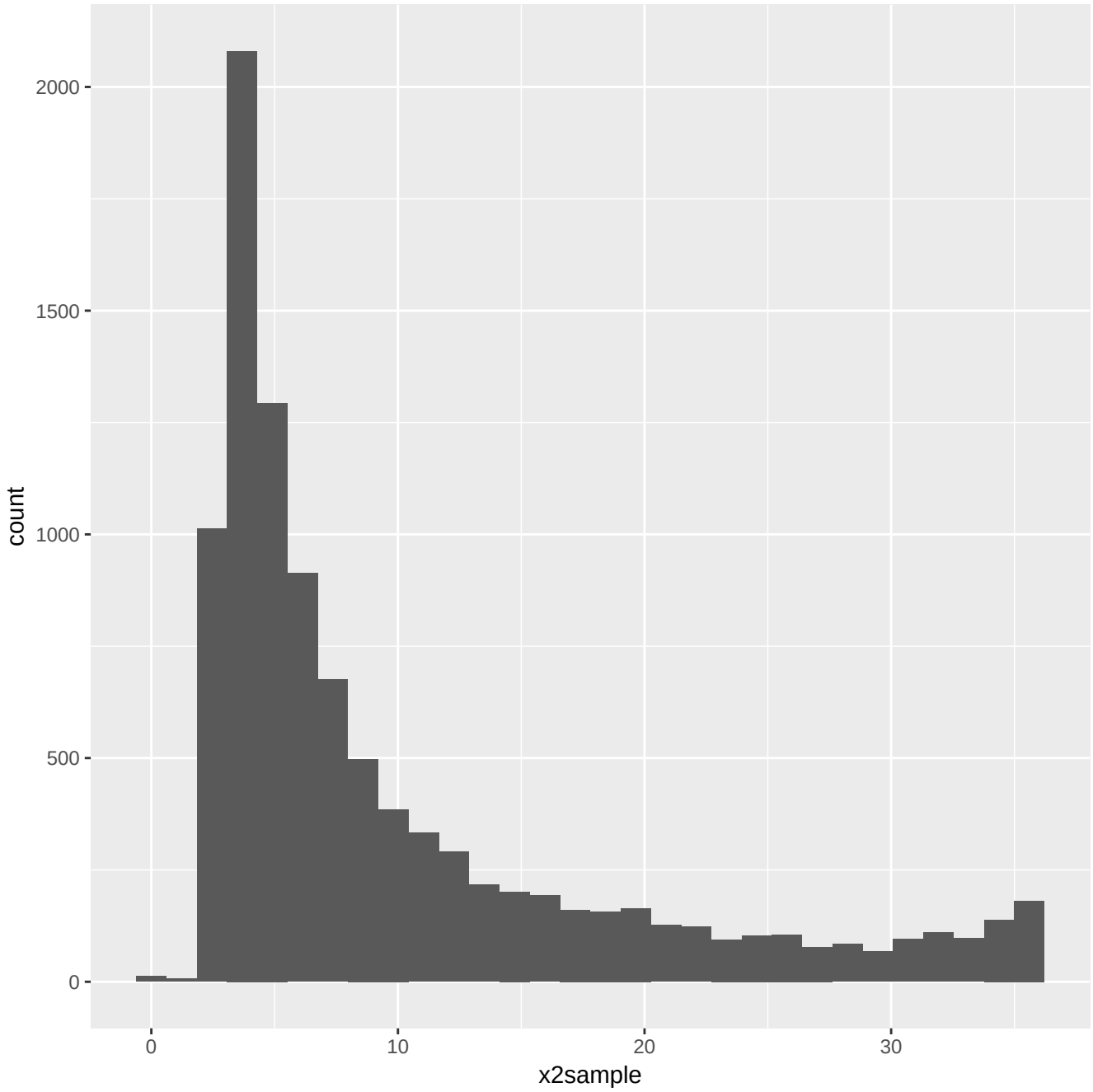
```
# Histogram of h(x)f(x)/g(x) and f(x)/g(x)
library(ggplot2)
qplot(xsample, geom = "histogram") # h(x) = x
```

```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```



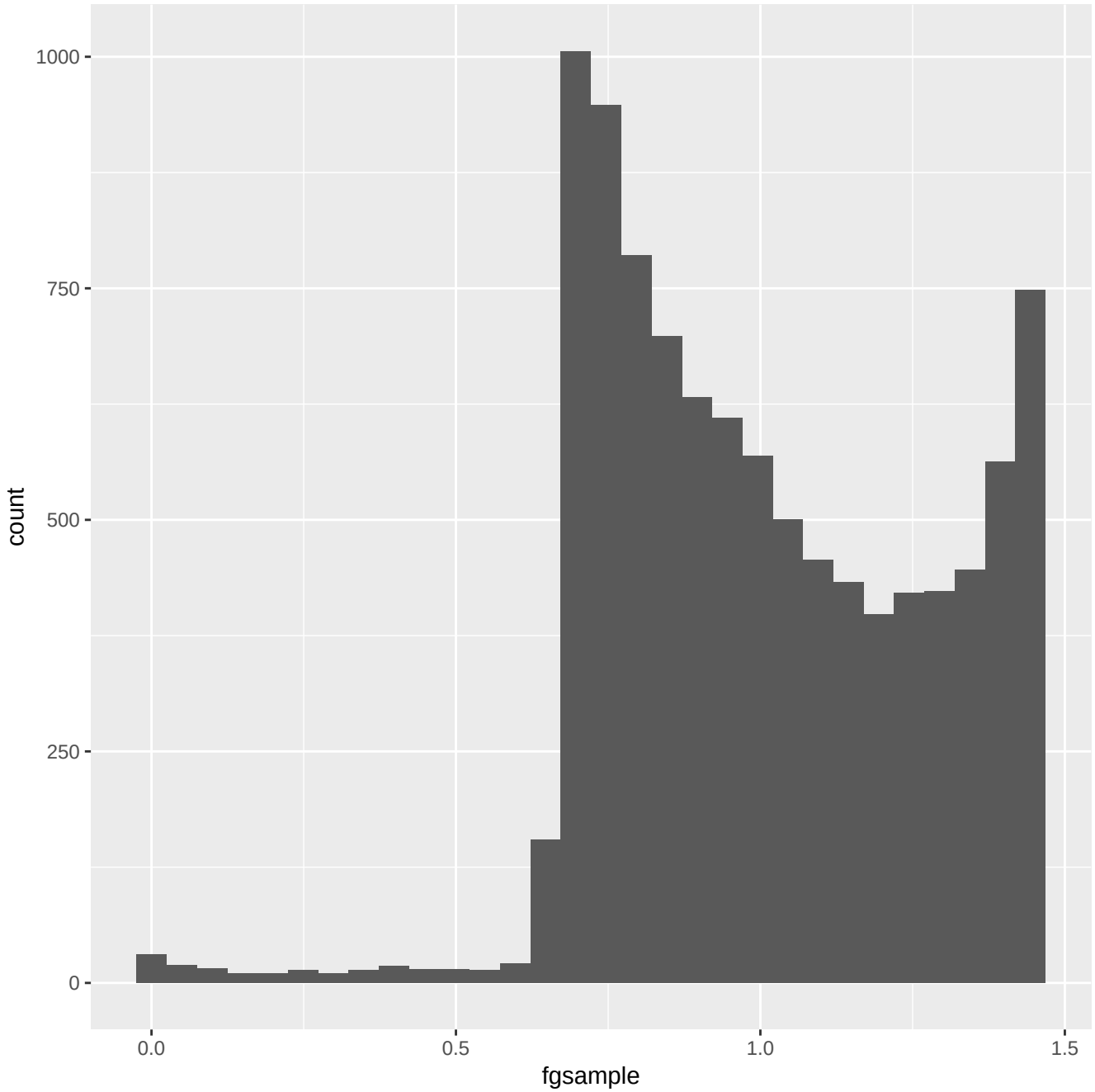

```
qplot(x2sample, geom = "histogram") # h(x) = x^2
```

```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```




```
qplot(fgsample, geom = "histogram")
```

```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```



Most of the weights lie in the interval (0.5, 1.5) and only few in (0, 0.5). There is not strong influence on estimated h(X).

1.(c)

```
# exchange f and g
for(i in 1 : m){
  x <- rexp(n = 1) + scale
  fgsample[i] <- g(x) / f(x)
  xsample[i] <- x * g(x) / f(x)
```

```
  x2sample[i] <- (x^2) * g(x) / f(x)
}
mean(xsample) # Should be close to 3
```

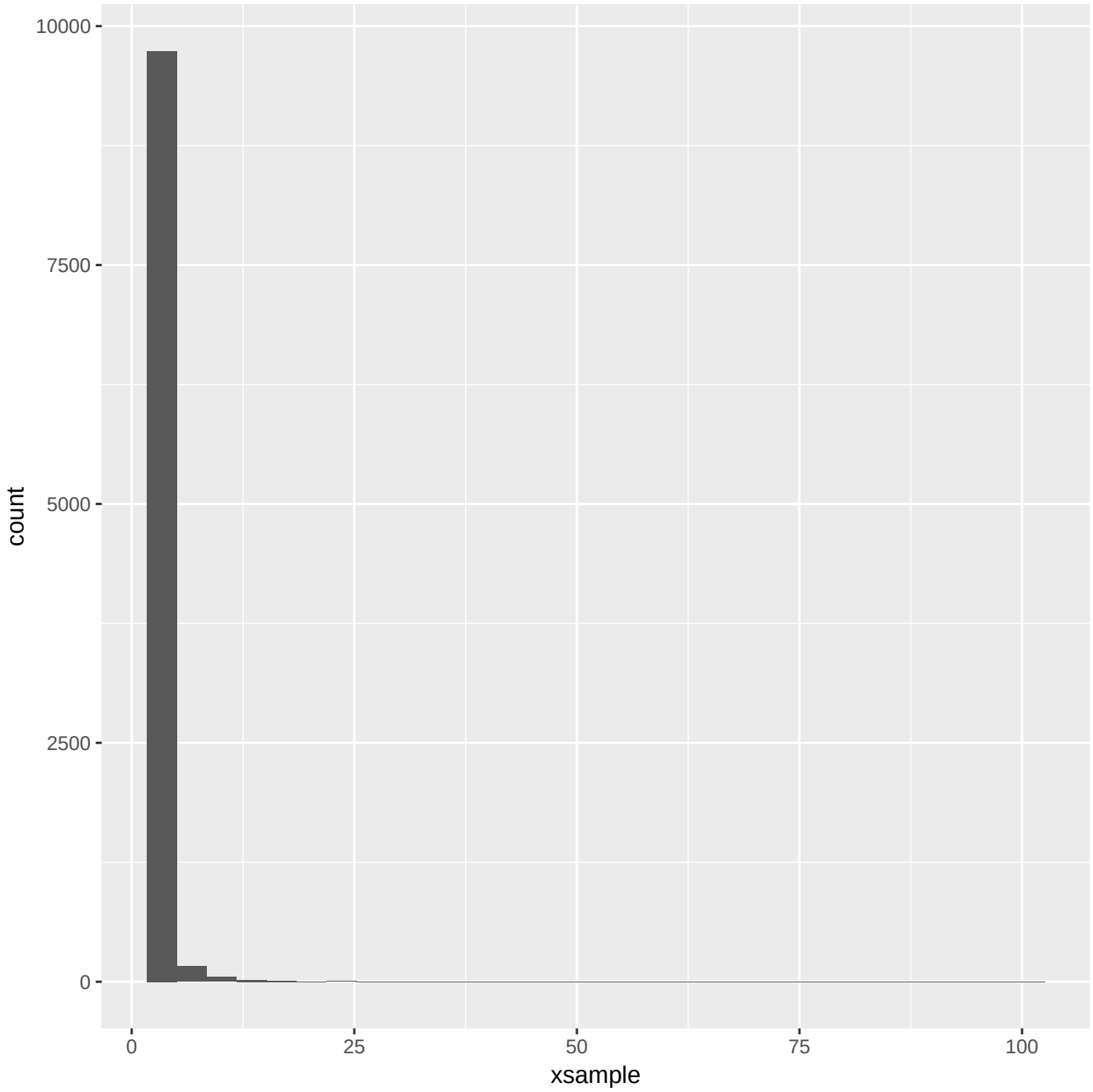
```
## [1] 2.88562
```

```
mean(x2sample) # Should be close to 12
```

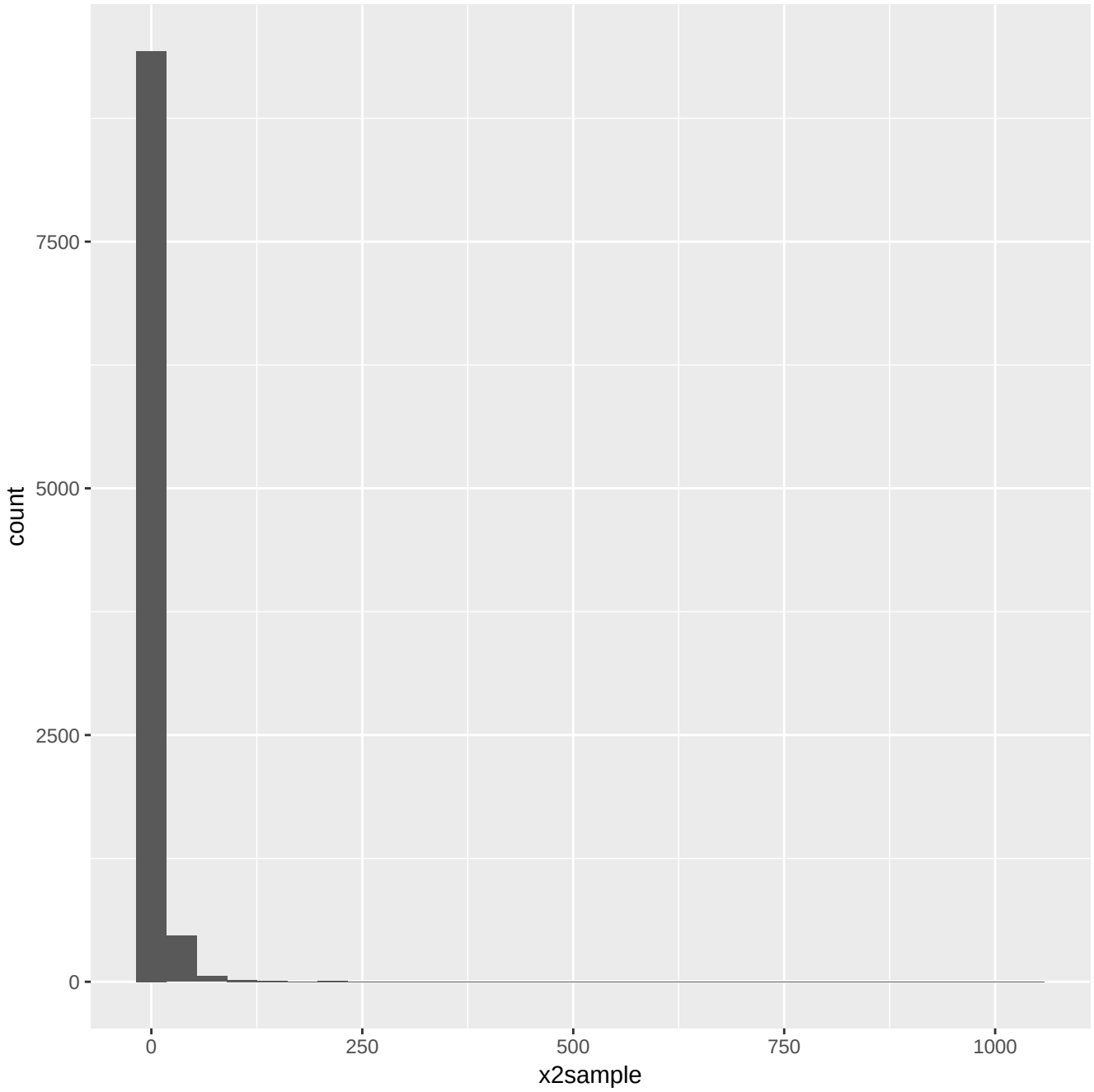
```
## [1] 9.672181
```

```
qplot(xsample, geom = "histogram") # h(x) = x
```

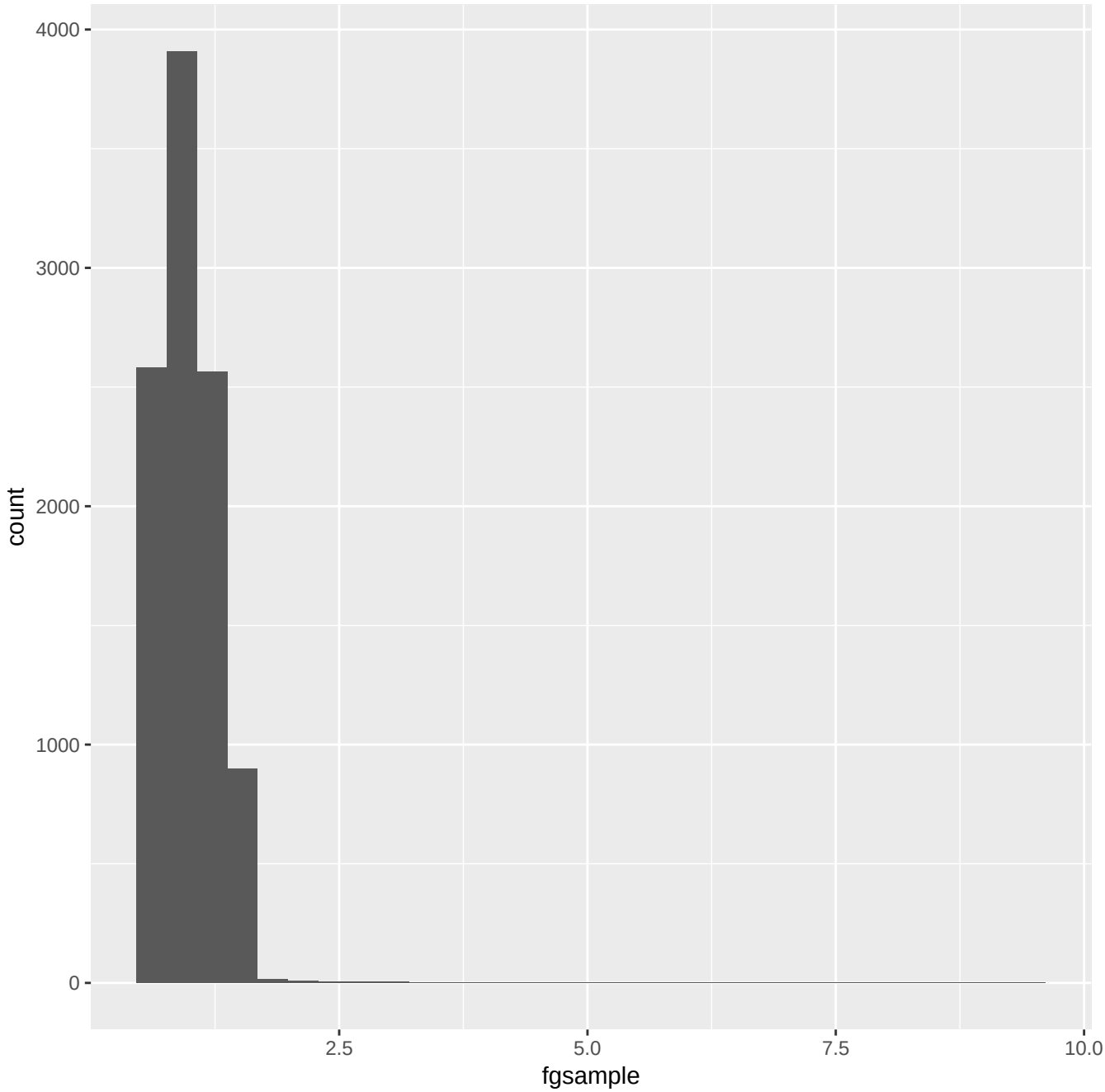
```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```




```
qplot(x2sample, geom = "histogram") # h(x) = x^2

## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```




```
qplot(fgsample, geom = "histogram")

## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```

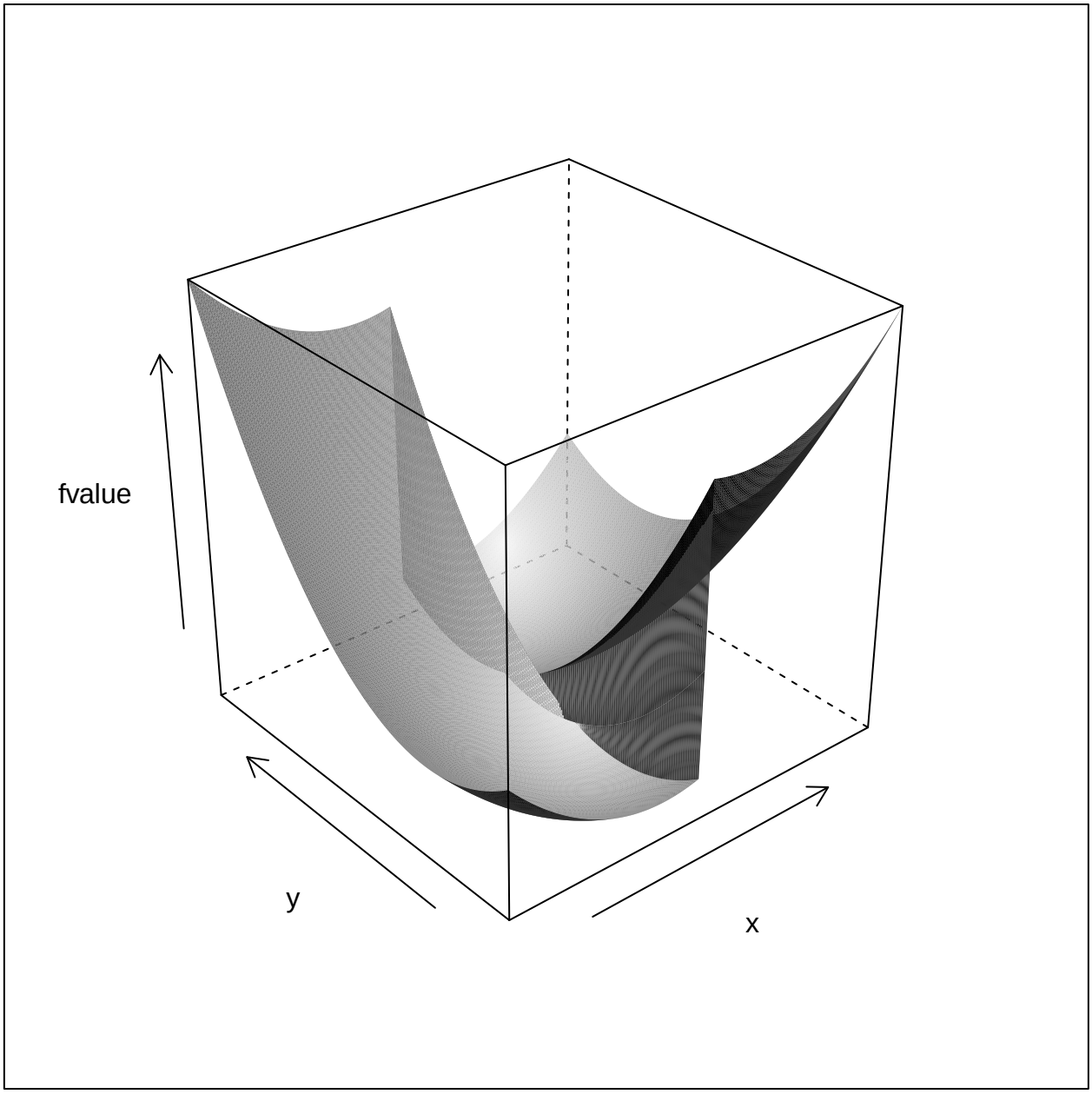


The histogram shows strong variance of the weights, which indirectly impacts estimated h(X). The strong variance comes from sampling from exponential distribution with fast-decaying tail(lighter tail).

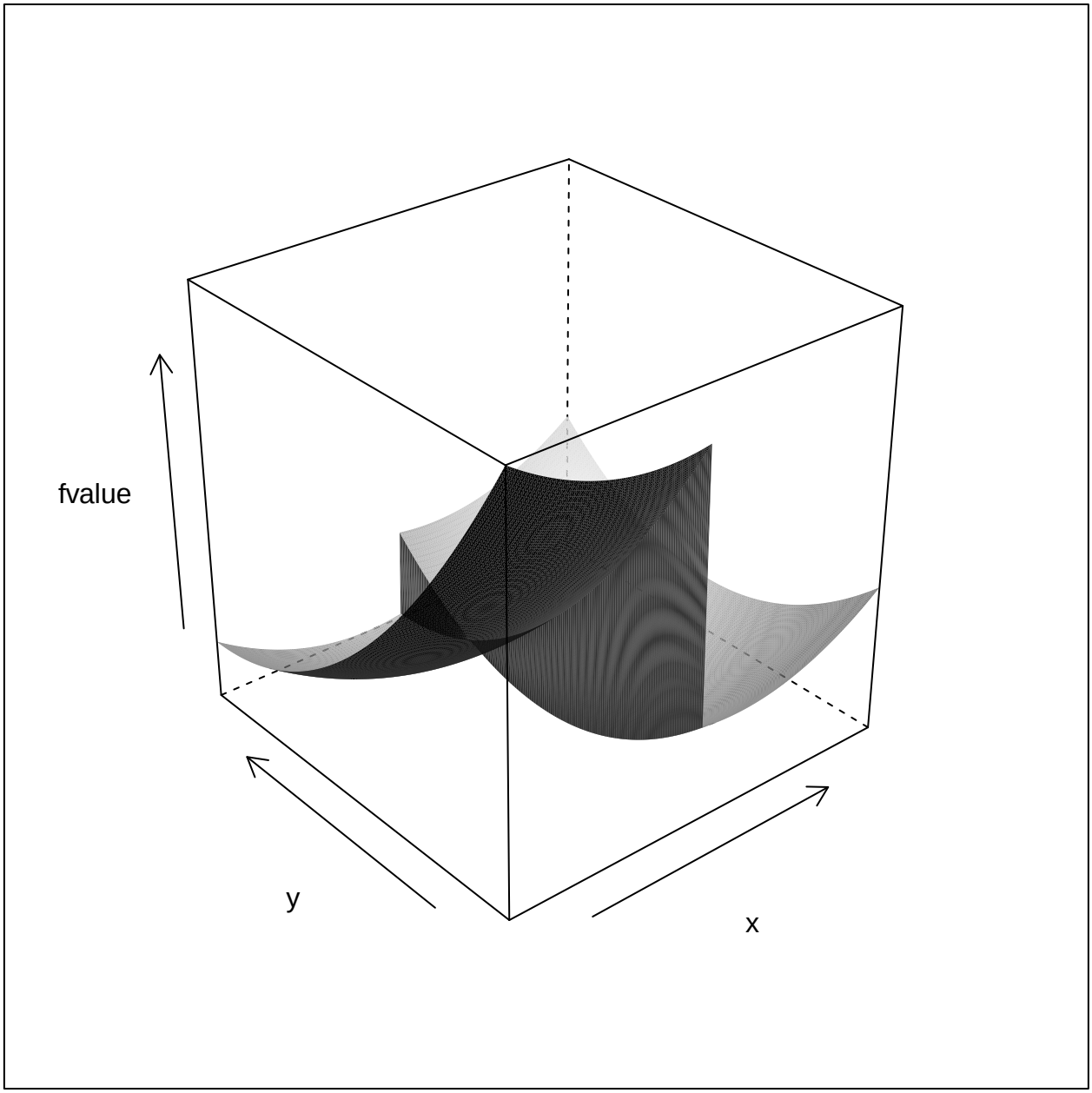
2.

```
# helical valley
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)

f <- function(x) {
```

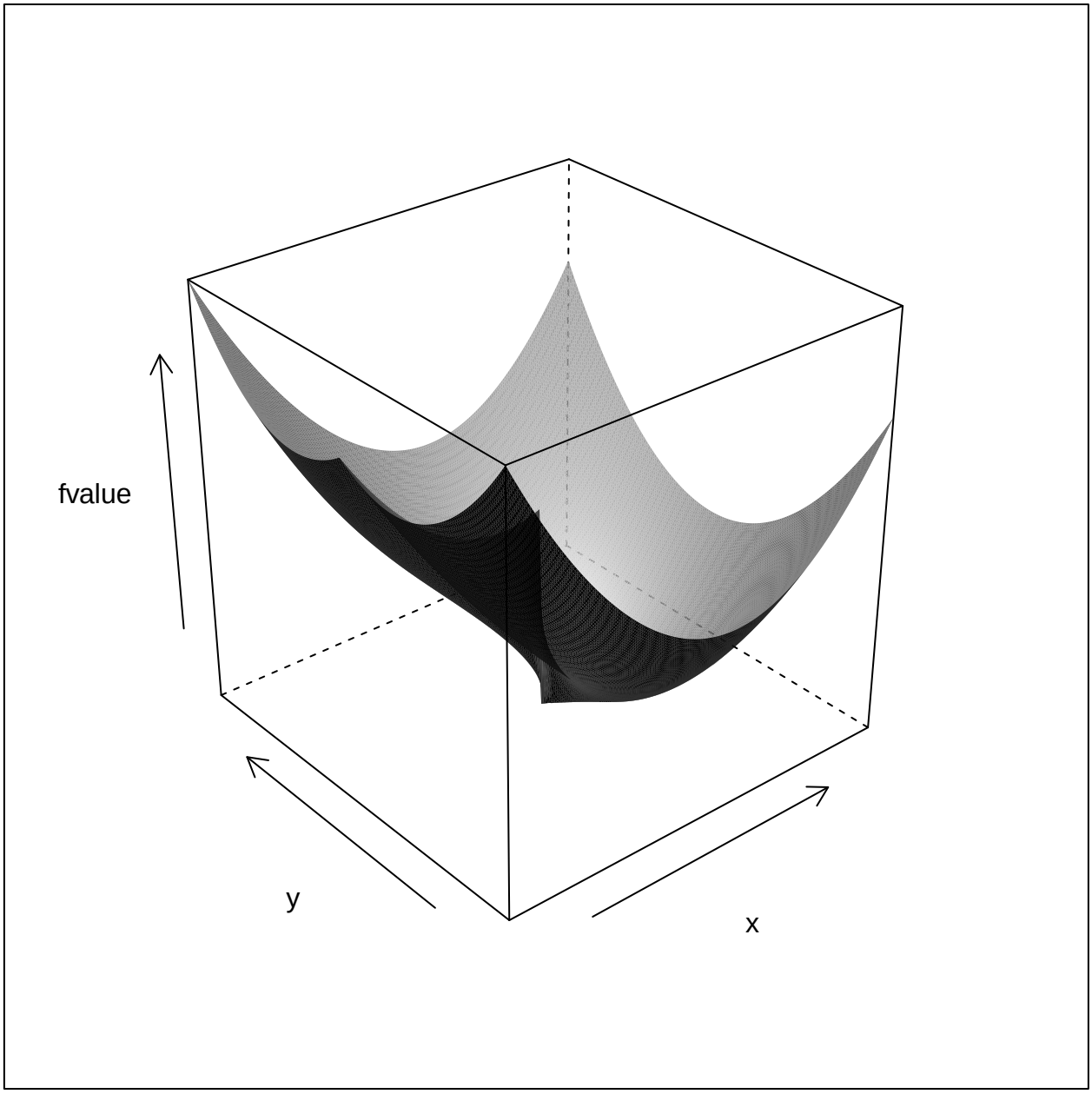
```
  f1 <- 10*(x[3] - 10*theta(x[1],x[2]))
  f2 <- 10*(sqrt(x[1]^2 + x[2]^2) - 1)
  f3 <- x[3]
  return(f1^2 + f2^2 + f3^2)
}
# plot 3D figure by lattice.
library(lattice)
myplot <- function(mydata){
  wireframe(fvalue ~ x*y, data = mydata, scales = list(col = "black"),shade =
  light.source = c(0,10,10), shade.colors = function(irr, ref, height, w = 0.4
}
# fix one variable, the other two range from -5 to 5
constant <- 0
n <- 100
L <- 2*n + 1 # -n ~ n
x <- vector(mode = "numeric", length = L^2)
y <- vector(mode = "numeric", length = L^2)
for(i in 1 : L){
  x[(L*(i-1)+1) : (L*i)] <- rep(5*(i-n-1)/n,L)
  y[(L*(i-1)+1) : (L*i)] <- seq(-5,5, length.out = L)
}

f1value <- vector(mode = "numeric", length = L^2) # fix x1
f2value <- vector(mode = "numeric", length = L^2) # fix x2
f3value <- vector(mode = "numeric", length = L^2) # fix x3
for(i in 1 : (L^2)){
     f1value[i] <- f(c(constant,x[i],y[i]))
     f2value[i] <- f(c(x[i],constant,y[i]))
     f3value[i] <- f(c(x[i],y[i],constant))
}
data1 <- data.frame(x,y,fvalue = f1value)
data2 <- data.frame(x,y,fvalue = f2value)
data3 <- data.frame(x,y,fvalue = f3value)
myplot(data1) # x means x2, y means x3
```




```
myplot(data2) # x means x1, y means x3
```




```
myplot(data3) # x means x1, y means x2
```



From the three figures we can confirm that : If one variable is fixed to 0, the function generally increases as the other two variables' absolute values increase to infinity. But near the center(the two variables are almost zero) there is an irregular trend.
Now we try to find the minimum by optim function.

```
# We try eight different starting points
start <- list(c(0,0,0),c(1,0,0),c(0,1,0),c(0,0,1),c(0,1,1),c(1,0,1),c(1,1,0),c
lowerbound <- c(-10,-10,-10)
result <- vector(mode = "list", length = 8)
for(i in 1 : 8){
    result[[i]] <- optim(par = start[[i]], fn = f, method = "L-BFGS-B",
```

```
}
result
```

```
## [[1]]
## [1] 0 0 0
##
## [[2]]
## [1] 1 0 0
##
## [[3]]
## [1]  1.000000e+00 -3.134707e-06 -6.369448e-06
##
## [[4]]
## [1]  1.000000e+00 -2.729993e-09 -4.597314e-09
##
## [[5]]
## [1] 1.000000e+00 1.027721e-07 1.548830e-07
##
## [[6]]
## [1] 1.000000e+00 3.148266e-07 5.006055e-07
##
## [[7]]
## [1] 1.000000e+00 4.130795e-08 5.962209e-08
##
## [[8]]
## [1] 1.000000e+00 3.939495e-07 5.773333e-07
```

When the start point is at (0,0,0), f(x) will reach a local minimum, which is (0,0,0). The other starting points will reach the global minimum (1,0,0).

3.(c)

```
set.seed(1)
n <- 100
# True values of parameters
beta0 <- 1
```

```r
beta1 <- 2
sigma2 <- 6

# Complete data
xComplete <- runif(n)
yComplete <- rnorm(n, beta0 + beta1*xComplete, sqrt(sigma2))
r <- rank(yComplete) # The order of yComplete
yComplete <- sort(yComplete, decreasing = FALSE)


# Now we rearrange data
xTemp <- vector(mode = "numeric", length = n)
for(i in 1 : n){ xTemp[r[i]] <- xComplete[i] }
xComplete <- xTemp

# Calculate regression consistents
myreg <- function(X,Y){
  mod <- lm(Y ~ X)
  ebeta0 <- summary(mod)$coef[1]
  ebeta1 <- summary(mod)$coef[2]
  meanresi <- mean((residuals.lm(mod))^2)
  return(c(ebeta0,ebeta1,meanresi))
}

# The relative difference of parameter between generation t and t+1
differ <- function(theta,nexttheta){
  return(sum(((nexttheta - theta) / theta)^2))
}

# Main function, with p = proportion of exceedance
myfun <- function(p){
  tau <- yComplete[n*(1-p)] # Threshold
  # Initialize
  Y <- yComplete[1:(n*(1-p))] # observed data
  X <- xComplete[1:(n*(1-p))] # corresponding X
  YZ <- yComplete[(n*(1-p)+1):n] # censored data
```

```r
XZ <- xComplete[(n*(1-p)+1):n] # corresponding X

# Calculate expected value and variance of truncated normal distribution
EV <- function(beta0,beta1,sigma2,x){
  mu <- beta0 + beta1*x
  tau2 <- (tau - mu) / sqrt(sigma2)
  rho <-  dnorm(tau2) / (1 - pnorm(tau2))
  E <- mu + sqrt(sigma2)*rho
  V <- sigma2*(1 + tau2*rho - (rho)^2)
  return(c(E,V))
}

EM <- function(beta0, beta1, sigma2){
# Regression of {Yc+1...Yn} on {Xc+1...Xn}
  Em <- vector(mode = "numeric", length = n*p)
  Vm <- vector(mode = "numeric", length = n*p)
  for(i in 1 : (n*p)){
    Em[i] <- EV(beta0, beta1, sigma2,XZ[i])[1]
    Vm[i] <- EV(beta0, beta1, sigma2,XZ[i])[2]
  }
  Ytemp <- c(Y,Em)
  nextbeta0 <- myreg(xComplete,Ytemp)[1]
  nextbeta1 <- myreg(xComplete,Ytemp)[2]
  nextsigma2 <- myreg(xComplete,Ytemp)[3] + sum(Vm)/n
  return(c(nextbeta0, nextbeta1, nextsigma2))
}

# Now start the EM algorithm
theta <- c(myreg(X,Y)[1],myreg(X,Y)[2],myreg(X,Y)[3]) # t = 0
# Do the EM interations until the difference is smaller than 0.001
crie <- 1
count <- 0
while(crie > 0.001){
  nexttheta <- EM(theta[1],theta[2],theta[3])
  crie <- differ(theta,nexttheta)
  theta <- nexttheta
```

```
    count <- count + 1
  }
  return(c(theta,count))
} # myFunction end
# Estimated beta0,beta1,sigma2 and iteration times
myfun(0.2)
```

```
## [1] 0.4563399 2.8158476 4.5870851 4.0000000
```

```
# Estimated beta0, beta1, sigma2 if no data are censored
myreg(xComplete, yComplete)
```

```
## [1] 0.5607442 2.7650812 5.2072910
```

Estimated parameters are close to those with no censored data. I set the stop criteria when the relative difference of the parameters between generations is smaller than 0.001, and it took 4 times.
Then, redo the code with higher proportion(p = 0.8)

```
myfun(0.8)
```

```
## [1]  0.2497301  2.2770912  3.0019199 24.0000000
```

Since more data are censored, it takes more time for convergence(24). Also, the estimated parameters are farther to those with no censored data.

3.(d)

```
# consider p = 0.8
p <- 0.8
tau <- yComplete[n*(1-p)] # Threshold
Y <- yComplete[1:(n*(1-p))] # observed data
X <- xComplete[1:(n*(1-p))] # corresponding X
YZ <- yComplete[(n*(1-p)+1):n] # censored data
XZ <- xComplete[(n*(1-p)+1):n] # corresponding X

# Minimize L
L <- function(theta){
  beta0 <- theta[1]
```

```
  beta1 <- theta[2]
  sigma2 <- theta[3]
  # Reparameterization
  mu <- rep(beta0, n) + beta1*xComplete
  y <- (Y - mu[1:(n*(1-p))]) / sqrt(sigma2)
  yz <- (tau - mu[((n*(1-p))+1):n]) / sqrt(sigma2)
  return(n*(1-p)*log(sqrt(sigma2)) + sum(y^2)/2 - sum(log(1 - pnorm(yz))))
}

theta <- c(myreg(X,Y)[1],myreg(X,Y)[2],myreg(X,Y)[3]) # start point
# Do the BFGS interations until the difference is smaller than 0.001
crie <- 1
count <- 0
while(crie > 0.001){
    nexttheta <- optim(par = c(theta[1],theta[2],theta[3]), fn = L, method = "
    crie <- differ(theta,nexttheta)
    theta <- nexttheta
    count <- count + 1
}
```

```
## Warning in sqrt(sigma2):   NaNs
## Warning in sqrt(sigma2):   NaNs
## Warning in sqrt(sigma2):   NaNs
```

```
theta
```

```
## [1] 0.3020583 2.6052337 3.5010535
```

```
count
```

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

The result shows that BFGS gives the closer estimation than EM algorithm and faster to reach convergence(only 2 times).