Stats243 Problem Set 8

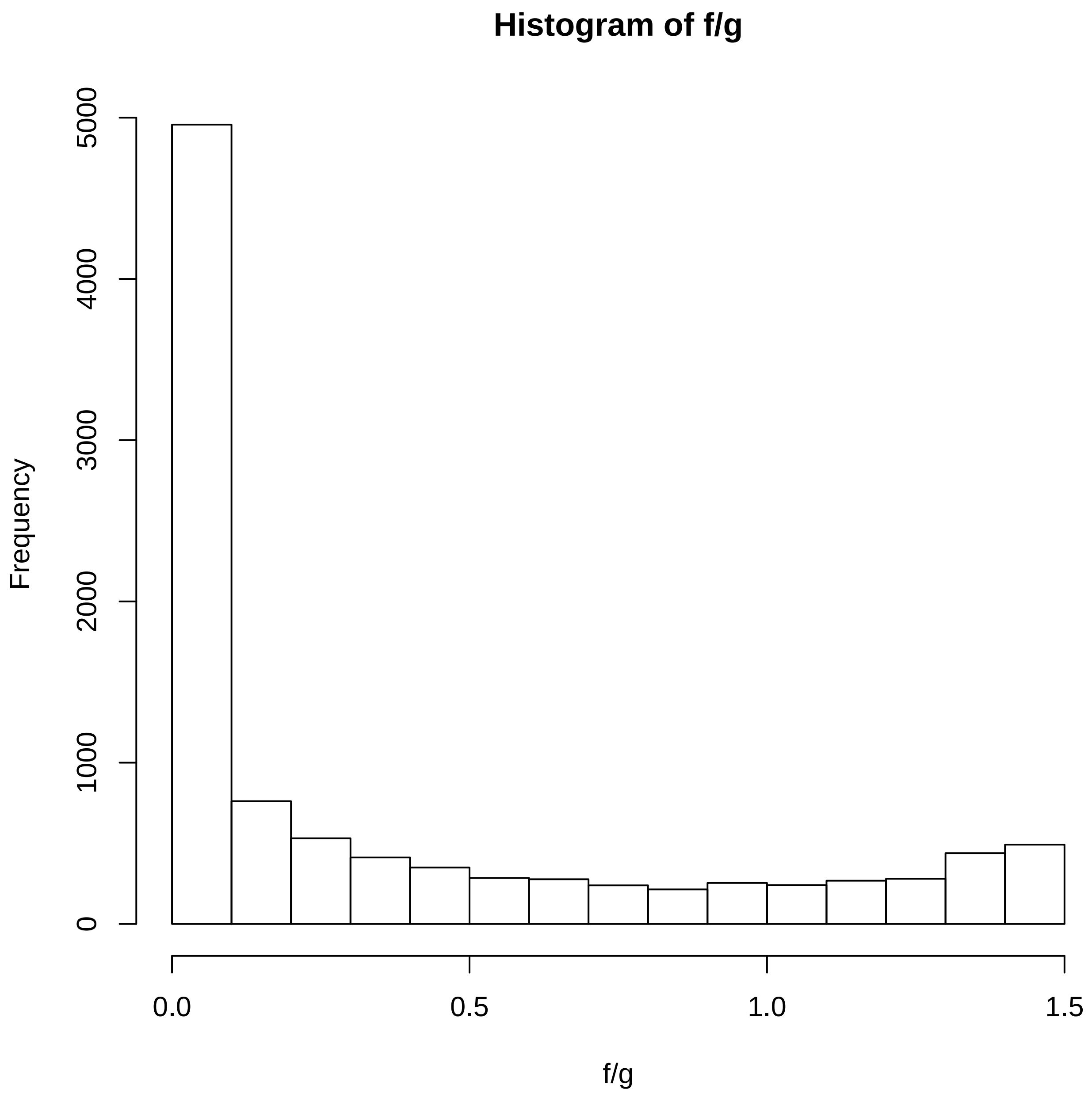
Mengling Liu

Nov 2017

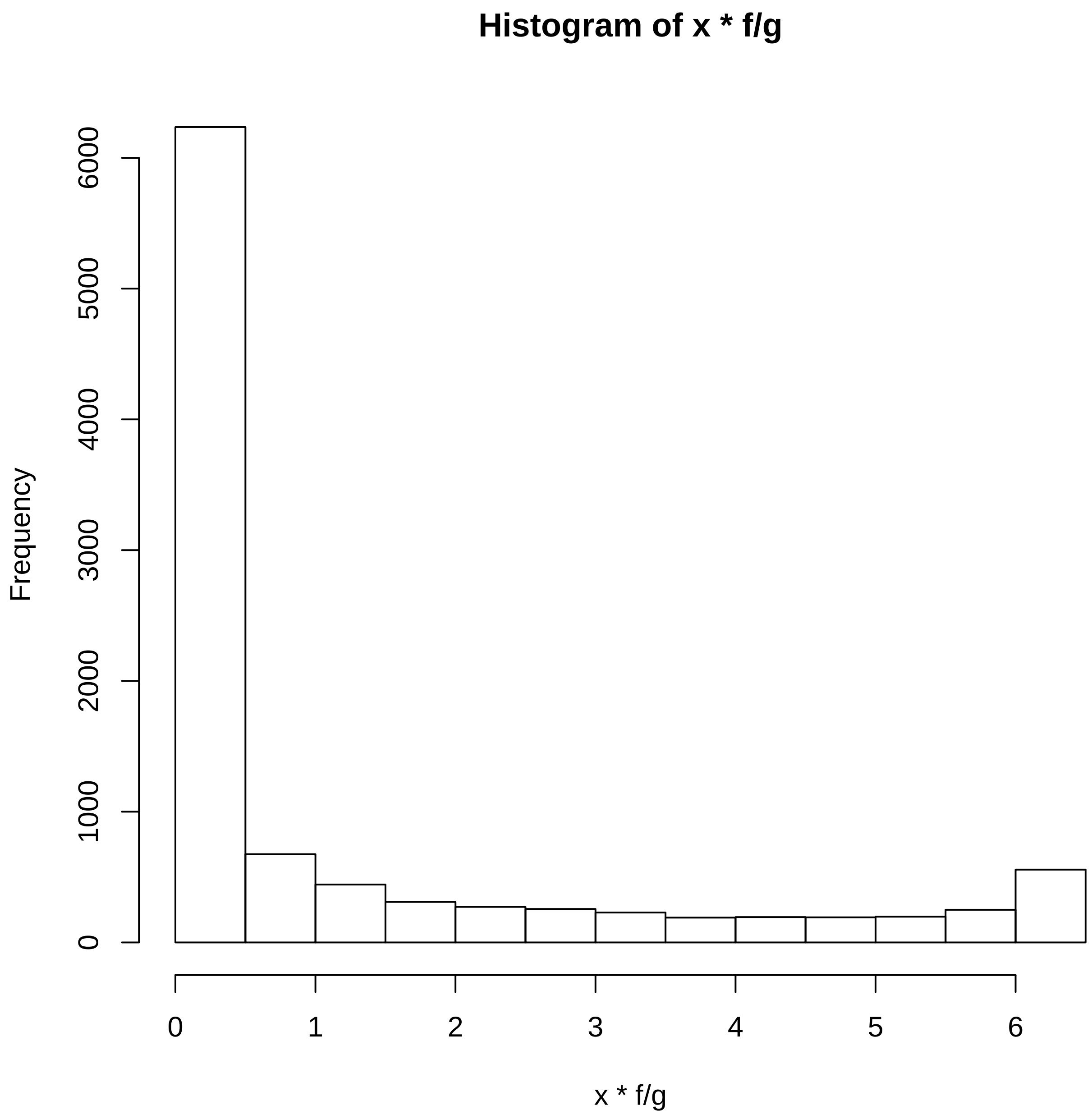
Collaboration: I disscussed with Rui Chen and Fan Dong for this homework set.

1. Importance sampling

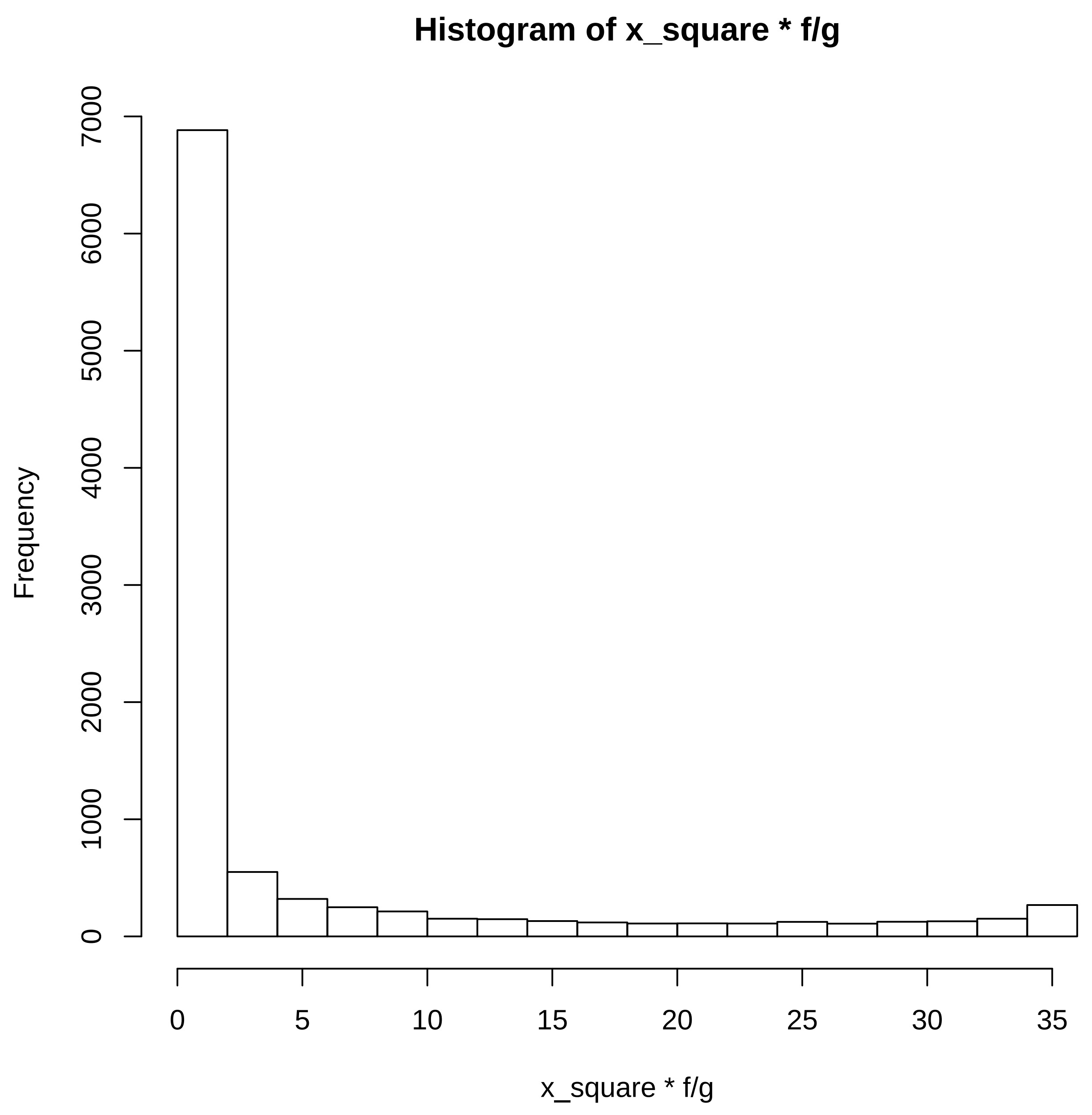
```
library(rmutil)

##
## Attaching package:  'rmutil'
## The following object is masked from 'package:stats':
##
##    nobs

# (b) Use m = 10000 to estimate EX and E(X^2)
# g(x) is Pareto distribution, f(x) is exponential distribution
# h(x) is x
alpha <- 2
beta <- 3
m<-10000
x<- rpareto(m,alpha,beta)
f <- exp(-(x-2))
g <- beta * (alpha^beta)/(x^(beta+1))
#histogram of weights
hist(f/g)
```



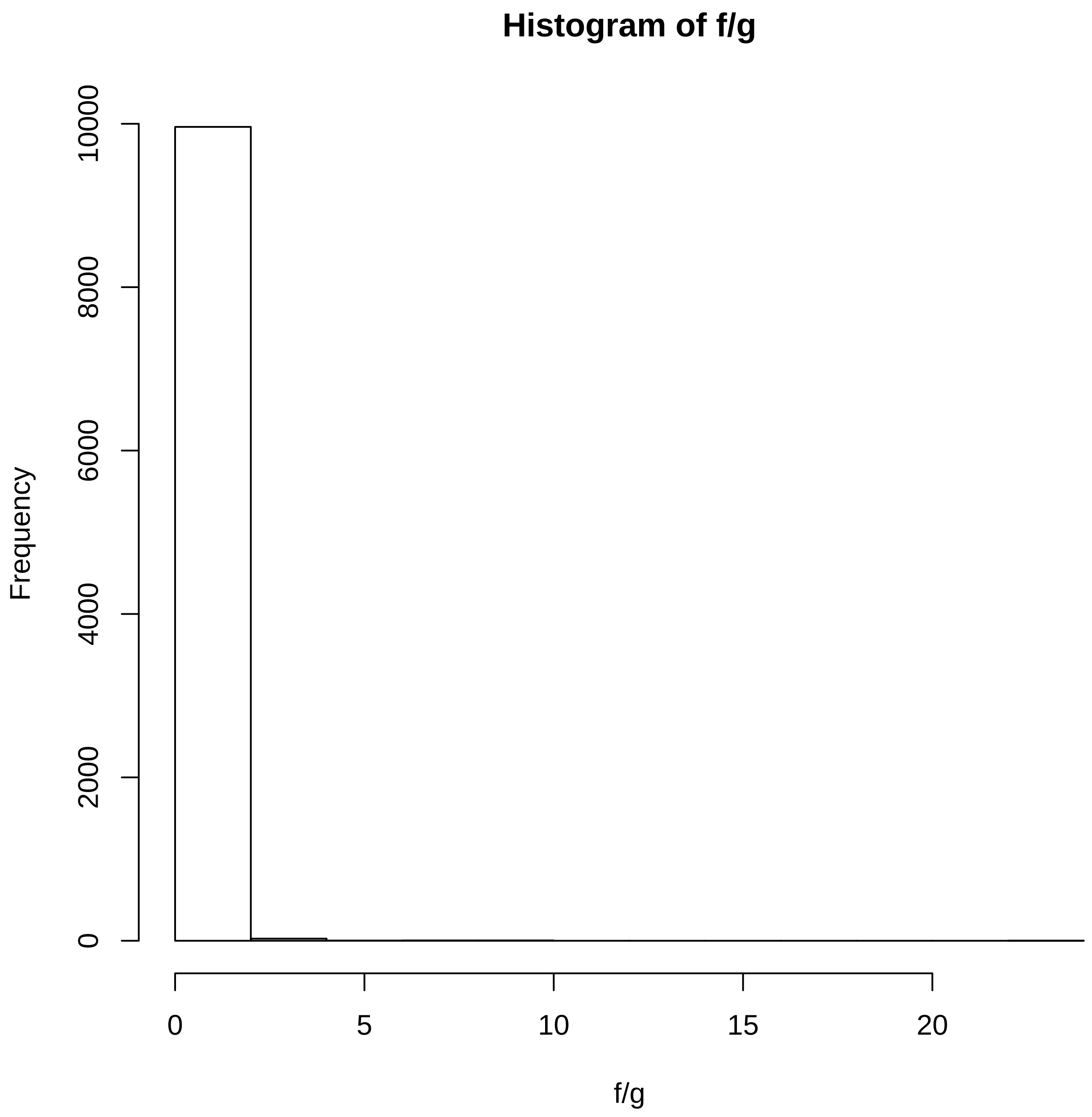
```
#histogram of h(x)*f(x)/g(x)
hist(x*f/g)
```



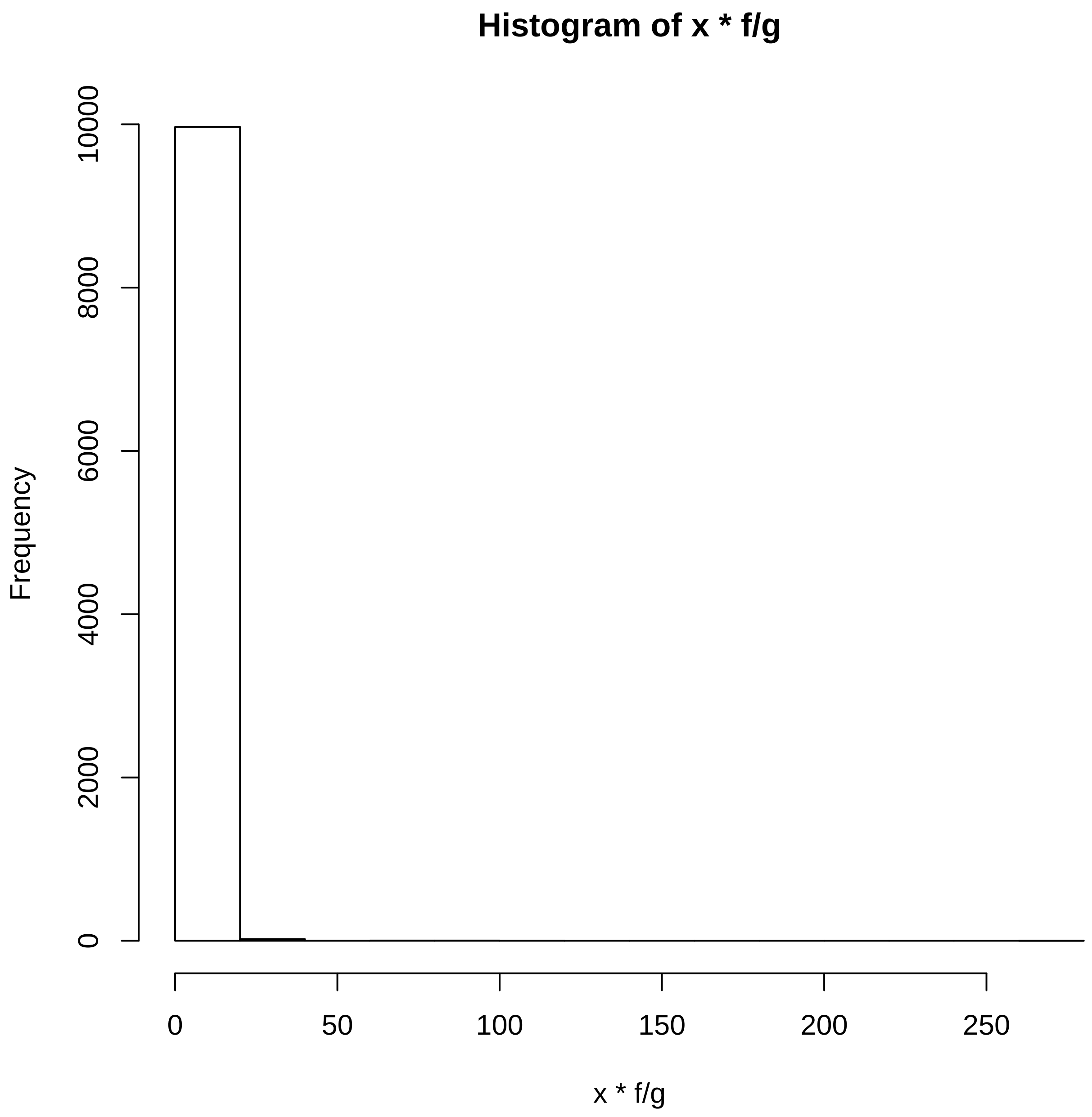
```
var(x*f/g)

## [1] 3.875359

#h(x) is x^2
x_square <- x^2
hist(x_square*f/g)
```



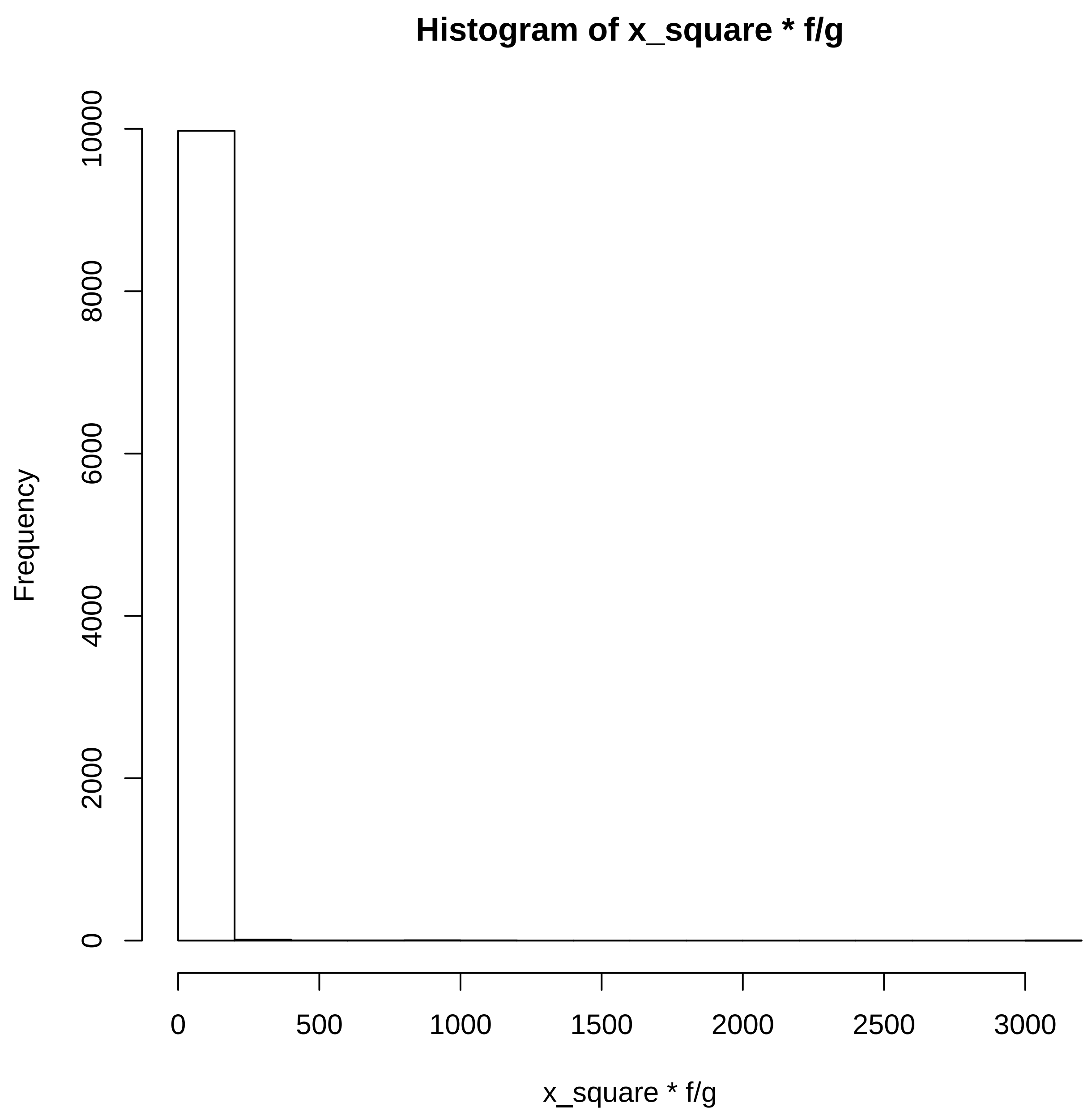
```
var(x_square*f/g)

## [1] 92.11892

## (c) g(x) is exponential distribution, f(x) is Pareto distribution
alpha <- 2
beta <- 3
m<-10000
x<- rexp(m,1)+2
g <- exp(-(x-2))
f <- beta * (alpha^beta)/(x^(beta+1))
#histogram of weights
hist(f/g)
```



```r
#histogram of h(x)*f(x)/g(x)
hist(x*f/g)
```



```
var(x*f/g)

## [1] 14.68021

#h(x) is x^2
x_square <- x^2
hist(x_square*f/g)
```

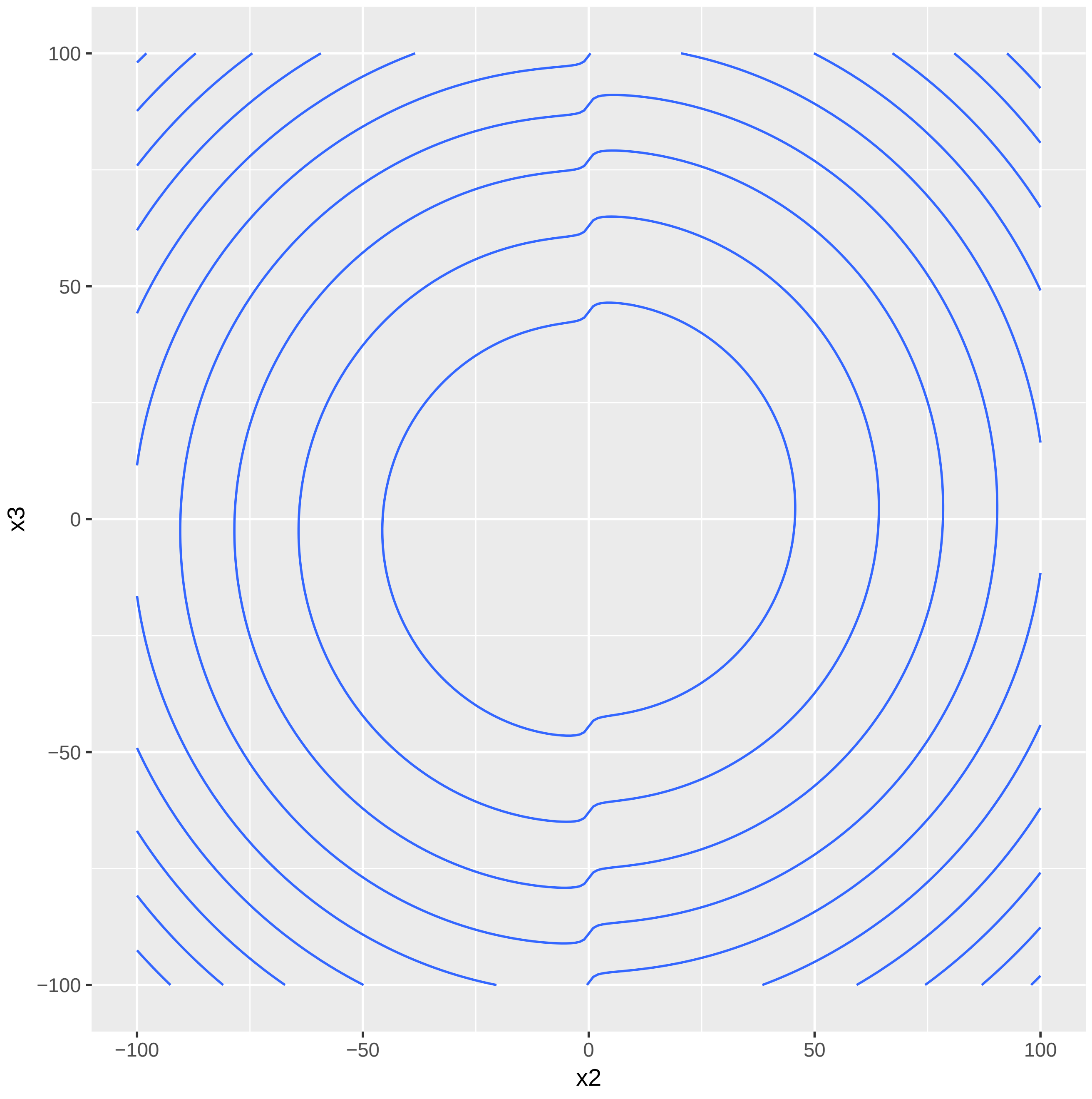


```
var(x_square*f/g)

## [1] 1759.284
```

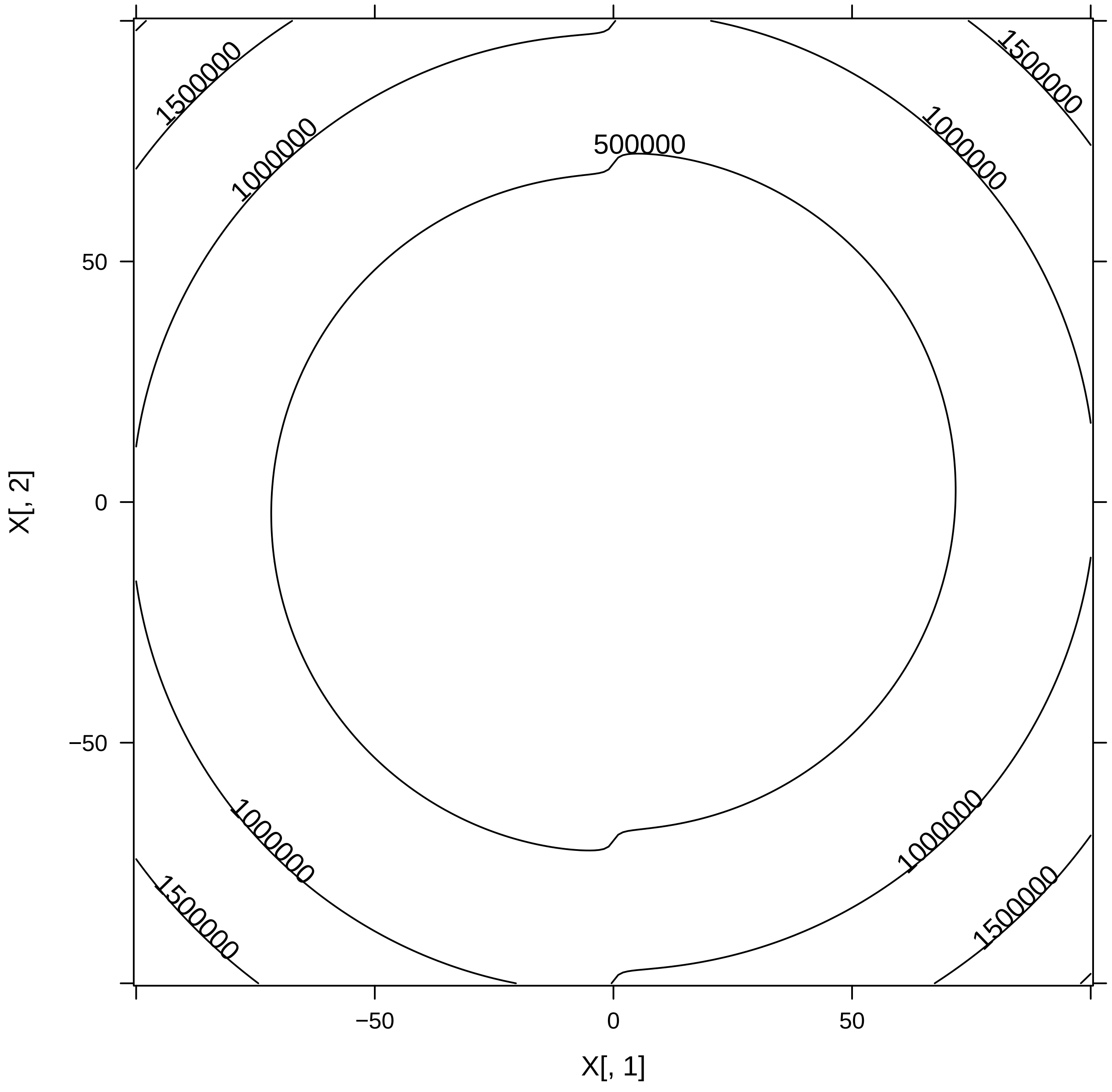
2. Helical valley

```
library(ggplot2)

## Warning:  package 'ggplot2' was built under R version 3.4.2

library(lattice)
x1<-1
x2<-c(-100:100)
x3<-c(-100:100)
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)
```

```
f <- function(x2,x3) {
  f1 <- 10*(x3 - 10*theta(x1,x2))
  f2 <- 10*(sqrt(x1^2 + x2^2) - 1)
  f3 <- x3
  return(f1^2 + f2^2 + f3^2)
}

X<-expand.grid(x2,x3)
result<-f(X[,1],X[,2])
data<-data.frame(result,X[,1],X[,2])
ggplot(data=data,aes(x=X[,1],y=X[,2],z=result))+geom_contour()+labs(x="x2",y="x3")
```



```
contourplot(result~X[,1] + X[,2], data=data)
```



```
# Finding the minimum of the function
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)
}

init<-c(1,1,1)
init<-c(1,1,0)
init<-c(100,100,100)
init<-c(-10000,10000,10000)
```

```r
optim(init,f,method="Nelder-Mead")

## $par
## [1]  3.5151144 -1.8360655  0.3013482
##
## $value
## [1] 993.6023
##
## $counts
## function gradient
##       98       NA
##
## $convergence
## [1] 0
##
## $message
## NULL

# There are mutiple local minimum generated
# One of the local minimum is
# x1=1, x2=0, x3=0, and the value of function equals zero
```

3(c). Censored regression problem

```r
set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

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

# a modest proportion of exceedances expected 20%
f1<-function(x,yComplete){
threshold<-sort(yComplete)[80]
x1<-x[which(yComplete<=threshold)]
y1<-yComplete[which(yComplete<=threshold)]
missingx1<-x[which(yComplete>threshold)]

mod1<-lm(y1~x1)
beta0_start<-summary(mod1)$coef[1,1]
beta1_start<-summary(mod1)$coef[2,1]
sigma_start<-var(y1)

beta0_old<-beta0_start
beta1_old<-beta1_start
sigma_old<-sigma_start

Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
y1_new<-c(y1,ExpectedZ_new)
x1_new<-c(x1,missingx1)
mod2<-lm(y1_new~x1_new)
summary(mod2)$coef
```

```
beta0_new<-summary(mod2)$coef[1,1]
beta1_new<-summary(mod2)$coef[2,1]
sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

count<-0
while (abs(beta0_new-beta0_old)>0.01 | abs(beta1_new-beta1_old)>0.01 | abs(sigma_new-sigma_old)>0.01){
  beta0_old<-beta0_new
  beta1_old<-beta1_new
  sigma_old<-sigma_new

  Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
  Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
  ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
  y1_new<-c(y1,ExpectedZ_new)
  x1_new<-c(x1,missingx1)
  mod2<-lm(y1_new~x1_new)
  summary(mod2)$coef

  beta0_new<-summary(mod2)$coef[1,1]
  beta1_new<-summary(mod2)$coef[2,1]
  sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

  count<-count+1
  print(count)
  print(sigma_new)
}
return(c(count=count,beta0_new=beta0_new,beta1_new=beta1_new,sigma_new=sigma_new))
}
f1(x,yComplete)

## [1] 1
## [1] 4.537895
## [1] 2
## [1] 4.586531
## [1] 3
## [1] 4.60258
## [1] 4
## [1] 4.607632
##     count beta0_new beta1_new sigma_new
## 4.0000000 0.4562646 2.8213764 4.6076318

# a modest proportion of exceedances expected 80%
f2<-function(x,yComplete){
threshold<-sort(yComplete)[20]
x1<-x[which(yComplete<=threshold)]
y1<-yComplete[which(yComplete<=threshold)]
missingx1<-x[which(yComplete>threshold)]

mod1<-lm(y1~x1)
beta0_start<-summary(mod1)$coef[1,1]
beta1_start<-summary(mod1)$coef[2,1]
sigma_start<-var(y1)

beta0_old<-beta0_start
beta1_old<-beta1_start
sigma_old<-sigma_start
```

```
Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
y1_new<-c(y1,ExpectedZ_new)
x1_new<-c(x1,missingx1)
mod2<-lm(y1_new~x1_new)
summary(mod2)$coef

beta0_new<-summary(mod2)$coef[1,1]
beta1_new<-summary(mod2)$coef[2,1]
sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

count<-0
while (abs(beta0_new-beta0_old)>0.01 | abs(beta1_new-beta1_old)>0.01 | abs(sigma_new-sigma_old)>0.01){
  beta0_old<-beta0_new
  beta1_old<-beta1_new
  sigma_old<-sigma_new

  Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
  Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
  ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
  y1_new<-c(y1,ExpectedZ_new)
  x1_new<-c(x1,missingx1)
  mod2<-lm(y1_new~x1_new)
  summary(mod2)$coef

  beta0_new<-summary(mod2)$coef[1,1]
  beta1_new<-summary(mod2)$coef[2,1]
  sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

  count<-count+1
  print(count)
  print(sigma_new)
}
return(c(count=count,beta0_new=beta0_new,beta1_new=beta1_new,sigma_new=sigma_new))
}
f2(x,yComplete)
```

```
## [1] 1
## [1] 2.086779
## [1] 2
## [1] 2.030646
## [1] 3
## [1] 2.099056
## [1] 4
## [1] 2.19268
## [1] 5
## [1] 2.288027
## [1] 6
## [1] 2.379477
## [1] 7
## [1] 2.465394
## [1] 8
## [1] 2.545376
## [1] 9
## [1] 2.619498
## [1] 10
## [1] 2.68803
## [1] 11
## [1] 2.751315
## [1] 12
## [1] 2.809713
## [1] 13
## [1] 2.863579
## [1] 14
## [1] 2.913253
## [1] 15
## [1] 2.959052
## [1] 16
## [1] 3.001274
## [1] 17
## [1] 3.040192
## [1] 18
## [1] 3.076063
## [1] 19
## [1] 3.109122
## [1] 20
## [1] 3.139587
## [1] 21
## [1] 3.167661
## [1] 22
## [1] 3.193529
## [1] 23
## [1] 3.217363
## [1] 24
## [1] 3.239323
## [1] 25
## [1] 3.259554
## [1] 26
## [1] 3.278192
## [1] 27
## [1] 3.295362
## [1] 28
## [1] 3.311179
## [1] 29
## [1] 3.325748
## [1] 30
## [1] 3.339169
## [1] 31
## [1] 3.351531
## [1] 32
## [1] 3.362917
## [1] 33
## [1] 3.373404
## [1] 34
## [1] 3.383063
##     count beta0_new beta1_new sigma_new
## 34.000000  0.215243  2.692390  3.383063

## General function including threshold percentage as input
function_para<-function(x,yComplete,thresh){
threshold<-sort(yComplete)[thresh]
```

```r
x1<-x[which(yComplete<=threshold)]
y1<-yComplete[which(yComplete<=threshold)]
missingx1<-x[which(yComplete>threshold)]

mod1<-lm(y1~x1)
beta0_start<-summary(mod1)$coef[1,1]
beta1_start<-summary(mod1)$coef[2,1]
sigma_start<-var(y1)

beta0_old<-beta0_start
beta1_old<-beta1_start
sigma_old<-sigma_start

Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
y1_new<-c(y1,ExpectedZ_new)
x1_new<-c(x1,missingx1)
mod2<-lm(y1_new~x1_new)
summary(mod2)$coef

beta0_new<-summary(mod2)$coef[1,1]
beta1_new<-summary(mod2)$coef[2,1]
sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

count<-0
while (abs(beta0_new-beta0_old)>0.01 | abs(beta1_new-beta1_old)>0.01 | abs(sigma_new-sigma_old)>0.01){
  beta0_old<-beta0_new
  beta1_old<-beta1_new
  sigma_old<-sigma_new

  Tau_star<-(threshold-beta0_old-beta1_old*missingx1)/sqrt(sigma_old)
  Rou_threshold<-dnorm(Tau_star)/(1-pnorm(Tau_star))
  ExpectedZ_new<-beta0_old+beta1_old*missingx1+sqrt(sigma_old)*Rou_threshold
  y1_new<-c(y1,ExpectedZ_new)
  x1_new<-c(x1,missingx1)
  mod2<-lm(y1_new~x1_new)
  summary(mod2)$coef

  beta0_new<-summary(mod2)$coef[1,1]
  beta1_new<-summary(mod2)$coef[2,1]
  sigma_new<-(sum((y1-beta0_old-beta1_old*x1)^2)+sum(sigma_old*(1+Tau_star*Rou_threshold)))/n

  count<-count+1
  # print(count)
  # print(beta0_new)
  # print(beta1_new)
  # print(sigma_new)
}
return(c(count=count,beta0_new=beta0_new,beta1_new=beta1_new,sigma_new=sigma_new))
}

function_para(x,yComplete,20)

##     count beta0_new beta1_new sigma_new
## 34.000000  0.215243  2.692390  3.383063
```

```
function_para(x,yComplete,80)

##     count beta0_new beta1_new sigma_new
## 4.0000000 0.4562646 2.8213764 4.6076318
```

3(d). Maximize log-likelihood of the observed data

```
## a modest proportion of exceedances expected 20%
threshold<-sort(yComplete)[80]
x1<-x[which(yComplete<=threshold)]
y1<-yComplete[which(yComplete<=threshold)]
missingx1<-x[which(yComplete>threshold)]
x1_new<-c(x1,missingx1)
m<-length(y1)
loglikelihood<-function(param){
  (m/2)*log(exp(param[3]))+(1/(2*(exp(param[3]))))*sum((y1-param[1]-param[2]*x1)^2)
  -sum(log(1-pnorm((threshold-param[1]-param[2]*missingx1)/sqrt(exp(param[3])))))
}

#initial value
mod1<-lm(y1~x1)
beta0_start2<-summary(mod1)$coef[1,1]
beta1_start2<-summary(mod1)$coef[2,1]
sigma_start2<-var(y1)
log_sigma<-log(sigma_start2)
param<-c(beta0_start2,beta1_start2,log_sigma)

# Find the local minimum of the loglikelihood function above
optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))

## $par
## [1] 23.935073 14.857933  1.274725
##
## $value
## [1] 0
##
## $counts
## function gradient
##        2        2
##
## $convergence
## [1] 0
##
## $message
## NULL

# Find the standard error of the estimated parameters
se<-exp(optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))$par[3])
se

## [1] 3.577716

# Find the number of iterations with initial points from b
count1<-optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))$count[2]
count1

## gradient
##        2
```

```r
## a modest proportion of exceedances expected 80% --------------------
threshold<-sort(yComplete)[20]
x1<-x[which(yComplete<=threshold)]
y1<-yComplete[which(yComplete<=threshold)]
missingx1<-x[which(yComplete>threshold)]
x1_new<-c(x1,missingx1)
m<-length(y1)
loglikelihood<-function(param){
  (m/2)*log(exp(param[3]))+(1/(2*(exp(param[3]))))*sum((y1-param[1]-param[2]*x1)^2)
  -sum(log(1-pnorm((threshold-param[1]-param[2]*missingx1)/sqrt(exp(param[3])))))
}

#initial value
mod1<-lm(y1~x1)
beta0_start2<-summary(mod1)$coef[1,1]
beta1_start2<-summary(mod1)$coef[2,1]
sigma_start2<-var(y1)
log_sigma<-log(sigma_start2)
param<-c(beta0_start2,beta1_start2,log_sigma)

# Find the local minimum of the loglikelihood function above
optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))

## $par
## [1] 15.6029373  8.3796608  0.8863901
##
## $value
## [1] 0
##
## $counts
## function gradient
##        2        2
##
## $convergence
## [1] 0
##
## $message
## NULL

# Find the standard error of the estimated parameters
se<-exp(optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))$par[3])
se

## [1] 2.426355

# Find the number of iterations with initial points from b
count2<-optim(c(1,1,1),loglikelihood,method="BFGS",control=list(parscale=c(1,1,0.1)))$count[2]
count2

## gradient
##        2

# Compare the counts of iterations with part b EM algorithm
# When 20 data missing, using optimum function will be require lower
# counts of iteraitons. When 80 data missing, using EM algorithm will require lower counts of iterations.
function_para(x,yComplete,80)[1]

## count
##     4
```

```
count1

## gradient
##        2

function_para(x,yComplete,20)[1]

## count
##    34

count2

## gradient
##        2
```