Problem Set 8

Yue Hu

Nov 2017

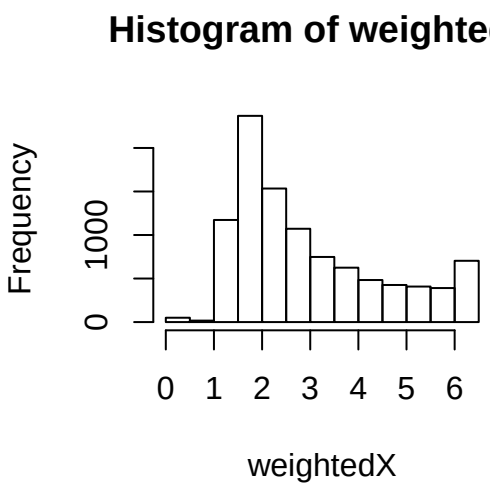
problem 1

(a)

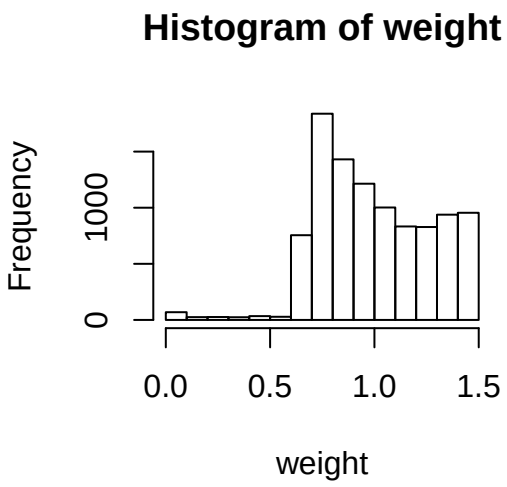
Parato distribution is a heavy-tailed distribution and its tail decays more slowly to 0.

(b)

```
library(EnvStats)
# generate sample
sample <- EnvStats::rpareto(10000, 2,3)

##calculate Ex
weightedX <- sapply(sample, function(x) {x*x^4*exp(2-x)/24})
EX <- mean(weightedX)
cat("Ex is", EX)

## Ex is 2.981199

hist(weightedX)
```




```
#histogram of weight
weight <- sapply(sample, function(x) {x^4*exp(2-x)/24})
hist(weight)
```




```
# calculate E(x^2)
weightedX2 <- sapply(sample, function(x) {x^2*x^4*exp(2-x)/24})
EX2 <- mean(weightedX2)
cat("E(x^2) is", EX2)

## E(x^2) is 9.883008

hist(weightedX2)
```

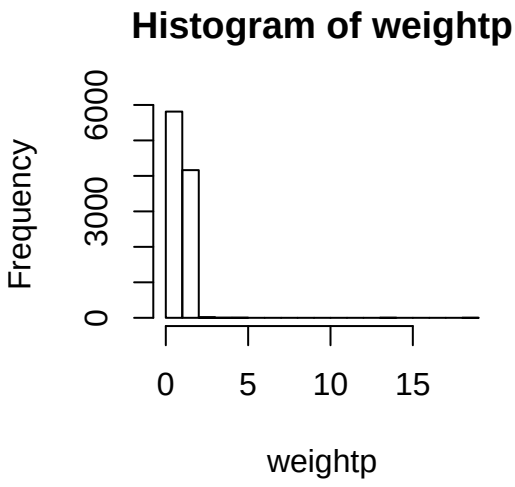
so we can see the value of EX and $E(X^2)$ is as expected.
And the variance is relatively small and resonable.

(c)

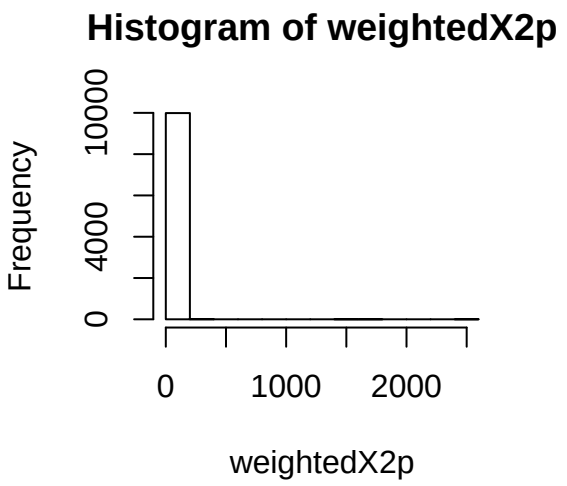
```
# generate sample
sample2 <- rexp(10000)
```

```r
sample2 <- sample2 +2

#calculate Ex
weightedXp <- sapply(sample2, function(x) {x*24/x^4/exp(2-x)})
EXp <- mean(weightedXp)
cat("EX is ", EXp)

## EX is  2.899011

hist(weightedXp)
```


```r
#histogram of weight
weightp <- sapply(sample2, function(x) {24/x^4/exp(2-x)})
hist(weightp)
```



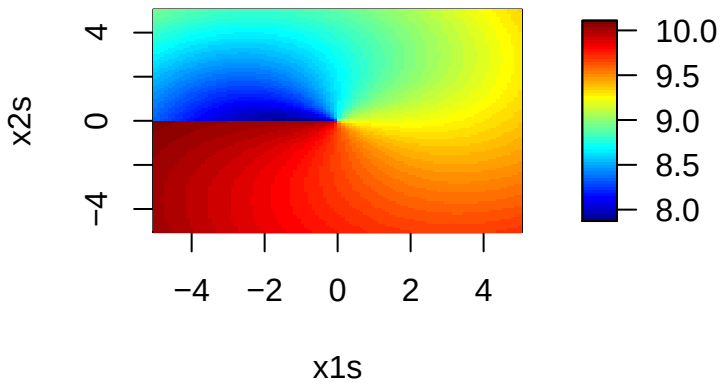
```r
#calculate E(X^2)
weightedX2p <- sapply(sample2, function(x) {x^2*24/x^4/exp(2-x)})
EX2p <- mean(weightedX2p)
cat("E(X^2) is ", EX2p)

## E(X^2) is  9.877829

hist(weightedX2p)
```



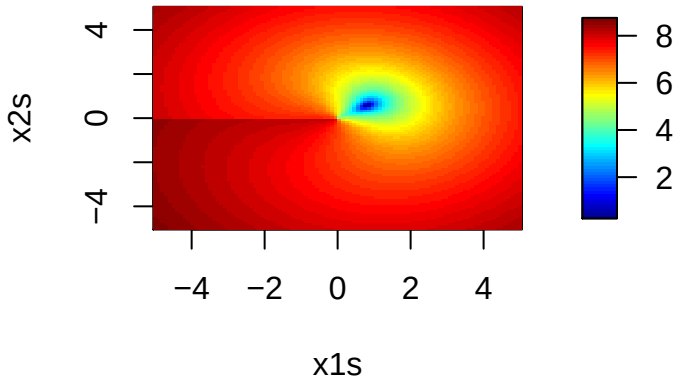
We can see the variance is much larger than previous case
IN fact,when g has thinner tails than f, f/g can be unbounded ,and the variance of EX can be very large.

problem 2

```r
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 slices
require(fields)
x1s <- seq(-5, 5, len = 100)
x2s = seq(-5, 5, len = 100)

#slice with x3 = 10
fx <- apply(cbind(expand.grid(x1s, x2s),10), 1, f)
image.plot(x1s, x2s, matrix(log(fx), 100, 100))
```




```
# slice with x3 = 1
fx2 <- apply(cbind(expand.grid(x1s, x2s),1), 1, f)
image.plot(x1s, x2s, matrix(log(fx2), 100, 100))
```




```
# optimize by optim()
init <- c(2,2,2)
optim(init, f, method = 'BFGS', control = list(trace = TRUE))

## initial  value 394.564575
## iter  10 value 2.282921
## iter  20 value 0.030005
## final  value 0.000000
## converged
## $par
## [1] 1.000000e+00 7.466597e-12 8.048624e-12
##
```

```
## $value
## [1] 4.587856e-20
##
## $counts
## function gradient
##       72       29
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
# use different starting points
init <- c(-200,20000,10000)
optim(init, f, method = 'BFGS', control = list(trace = TRUE))
```

```
## initial  value 50094968703.060165
## iter  10 value 675.661422
## iter  20 value 0.000653
## final  value 0.000000
## converged
## $par
## [1] 1.000000e+00 4.143270e-10 6.504327e-10
##
## $value
## [1] 4.320929e-19
##
## $counts
## function gradient
##       70       29
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
init <- c(2,2, 2)
nlm(f,init)
```

```
## $minimum
## [1] 1.70091e-08
##
## $estimate
## [1]  9.999995e-01 -8.223157e-05 -1.300799e-04
##
## $gradient
## [1] -1.343915e-08  1.545268e-08 -9.398712e-09
##
## $code
## [1] 1
##
## $iterations
## [1] 22
```

```
init <- c(2e16,2e16, -2e16)
nlm(f,init)

## $minimum
## [1] 1.700873e-08
##
## $estimate
## [1]  9.999995e-01 -8.223072e-05 -1.300783e-04
##
## $gradient
## [1] -1.914502e-08 -7.915570e-08  5.333774e-08
##
## $code
## [1] 1
##
## $iterations
## [1] 31
```

so we can see that the optimum parameter for f is (1,0,0). With different starting points they roughtly stop at the same point, though some are reaching better minimums.

Problem3

(a)

Set the value of censored data as Zi (i=1,2...c). And the rest value Yj (j=1,2...n-c) are known.And the liklihood should be

$$P(Y_i, Z_i|X;\theta) = \prod_i (2\pi\sigma^2)^{-1/2} e^{-\frac{(Z_i-\beta_0-\beta_1 x_i)^2}{2\sigma^2}} \prod_j (2\pi\sigma^2)^{-1/2} e^{-\frac{(y_j-\beta_0-\beta_1 x_j)^2}{2\sigma^2}}$$

and The log-liklihood shsould be

$$LL(Y_i, Z_i|X;\theta) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (Z_i-\beta_0-\beta_1 x_i)^2 - \frac{1}{2\sigma^2}\sum_j (Y_i-\beta_0-\beta_1 x_j)^2$$

For the E step, calculate

$$\begin{aligned}
Q &= E(Y_i, Z_i|X;\theta) \\
&= E[-\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (Z_i-\beta_0-\beta_1 x_i)^2 - \frac{1}{2\sigma^2}\sum_j (Y_i-\beta_0-\beta_1 x_j)^2] \\
&= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_j (Y_i-\beta_0-\beta_1 x_j)^2 - \frac{1}{2\sigma^2}\sum_i E[(Z_i-\beta_0-\beta_1 x_i)^2] \\
&= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_j (Y_i-\beta_0-\beta_1 x_j)^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{c} [(E(Z_i)-\beta_0-\beta_1 x_i)^2 + Var(Zi)] \\
&= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=i}^{n-c} (Y_i-\beta_0-\beta_1 x_j)^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{c} (E(Z_i)-\beta_0-\beta_1 x_i)^2 + cVar(Zi)
\end{aligned}$$

where $E(Z_i)$ and $Var(Z_i)$ is the expectation and variance of Zi based on the current value at iteration t, and should be comstant.

For M step, optimization is with respect to β_0, β_1, σ.

This is similar to linear regression and result should be

$$\hat{\beta}_0 = \frac{\sum_i (x_i - \bar{x_i})(Y_i - \bar{Y})}{\sum_i (x_i - \bar{x})^2}$$
$$\hat{\beta}_1 = \bar{Y} - \hat{\beta}_0 \bar{x}$$
$$\hat{\sigma^2} = \frac{\sum_{j=i}^{n-c} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_j)^2 + c(E_z - \hat{\beta}_0 - \hat{\beta}_1 x_j)^2 + cVar(Z)}{n}$$

Where Yi is the uncensored data plus c censored data with values E(z)

(b)

For initial value we can we can ignore datas where $y > \tau$, and use lm on the truncated y sequence.

(c)

For the censored data we can use $E(W|W > \tau)$ for y values, then use lm on the new y sequence to get betas. For sigma, the first two terms is the σ^2 by lm, and we need only add V(z)/n to it and take the sqare root.

We should break the optimization if objective function ceases to change.In this case, when the previous state and the current state considered by the EM algorithm are not significantly large.

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

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


# censor the sequence, 20% exceedance
tau1 <- quantile(yComplete, .8)
c <- 20

# calculate initial value

yTruncate <- yComplete
yTruncate[yTruncate>tau1] <- NA
xTruncate <- x
xTruncate[yTruncate>tau1] <- NA
mod <- lm(yTruncate~xTruncate)
betai <- summary(mod)$coef
sigmai <- summary(mod)$sigma

##
step = 0
for (i in 1:1000){
  # calculate Expectation and variance of Z based on current theta
  ymean <- betai[1]+betai[2]*mean(x)
  tauStar <- (tau1-ymean)/sigmai
  thou <- dnorm(tauStar)/(1-pnorm(tauStar))
  EZ <- ymean + sigmai* thou
```

```
  varZ <- sigmai^2*(1+tauStar*thou-thou^2)

  # construct new y sequence
  yNew <- yComplete
  yNew[yNew>tau1] <- EZ
  mod <- lm(yNew~x)
  # break the optimization if the difference between new and old parameters are small enough
  if (abs(summary(mod)$coef[1]-betai[1]) < 1e-16 & abs(summary(mod)$coef[2]-betai[2])< 1e-16)
  {break}
  # update theta
  betai <- summary(mod)$coef
  sigmai <-sqrt(summary(mod)$sigma + c*varZ/n)
  step <- step + 1

}

cat("theta is {" ,c(betai[1],betai[2], sigmai), "}, taking", step, "steps")

## theta is { 0.4855346 2.534128 1.421644 }, taking 17 steps
```

for sequence of 80% exceedance

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

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

# censor the sequence, 80%exceedance
tau2 <- quantile(yComplete, 0.2)
c <- 80
# calculate initial value

yTruncate <- yComplete
yTruncate[yTruncate>tau2] <- NA
xTruncate <- x
xTruncate[yTruncate>tau2] <- NA
mod <- lm(yTruncate~xTruncate)
betai <- summary(mod)$coef
sigmai <- summary(mod)$sigma

##
step = 0
for (i in 1:1000){
  yEstimate <- sapply(x, function(x) betai[1]+betai[2]*x)
  mean <- mean(yEstimate)
  tauStar <- (tau2-mean)/sigmai
  thou <- dnorm(tauStar)/(1-pnorm(tauStar))
  EZ <- mean + sigmai* thou
  varZ <- sigmai^2*(1+tauStar*thou-thou^2)
```

```
  # construct new y sequence
  yNew <- yComplete
  yNew[yNew>tau2] <- EZ
  mod <- lm(yNew~x)
  # break the optimization if the difference between new and old parameters are small enough
  if (abs(summary(mod)$coef[1]-betai[1]) < 1e-16 & abs(summary(mod)$coef[2]-betai[2])< 1e-16)
  {break}
  betai <- summary(mod)$coef
  sigmai <-sqrt(summary(mod)$sigma + c*varZ/n)
  step <- step + 1

}


cat("theta is {" ,c(betai[1],betai[2], sigmai), "}, taking", step, "steps")

## theta is { 0.4725494 1.253678 1.458363 }, taking 131 steps
```

(d)

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

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

# generate truncated sequence, 20%exceedance
tau1 <- quantile(yComplete, .8)

# construct truncated sequece
yTruncate <- yComplete
xTruncate <- x
xTruncate <- xTruncate[yTruncate < tau1]
yTruncate <- yTruncate[yTruncate < tau1]
n<- length(xTruncate)


# theta being {beta0, beta1, log(sigma) }
LL <- function(theta, x, y){
  yEstimate <- sapply (x, function(x) theta[1]+theta[2]*x)
  rss <-sum((yEstimate-y)^2)
  return(-n/2*log(2*3.14)-n*theta[3]-0.5*exp(-2*theta[3])*rss)
}

optim(c(0,0,0), LL, y = yTruncate,x = xTruncate, method = 'BFGS', control = list(trace = TRUE, parscale

## initial  value -233.200173
## final  value -237704919886603683466080282660666206424480828880482820486420882208244084600204224262404
```

```
## converged
## $par
## [1] -206.6395 -116.0925 -185.2685
##
## $value
## [1] -2.377049e+167
##
## $counts
## function gradient
##      376        4
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
# try several initial points to avoid local minima
optim(c(1.5,1.5,10), LL, y = yTruncate,x = xTruncate, method = 'BFGS', control = list(trace = TRUE, pars
```

```
## initial  value -873.494800
## final  value -1188422437680323420584 28.000000
## converged
## $par
## [1] 2.814750e+11 2.814750e+11 1.485528e+21
##
## $value
## [1] -1.188422e+23
##
## $counts
## function gradient
##        7        7
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optim(c(1,2,0.4), LL, y = yTruncate,x = xTruncate, method = 'BFGS', control = list(trace = TRUE, parsca
```

```
## initial  value -165.483237
## final  value -695395197143752898944820864846602660682268468802208820264840624420064226446888844884461
## converged
## $par
## [1]  684.7854  356.0316 -268.6741
##
## $value
## [1] -6.953952e+240
##
## $counts
## function gradient
##      366        3
##
## $convergence
```

```
## [1] 0
##
## $message
## NULL
```

```
optim(c(1,2,10), LL, y = yTruncate,x = xTruncate, method = 'BFGS', control = list(trace = TRUE, parscal
```

```
## initial  value -873.494799
## final  value -873.494799
## converged
## $par
## [1]  1  2 10
##
## $value
## [1] -873.4948
##
## $counts
## function gradient
##        2        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

So we can see that with theta = 1,2 ,10, the log-liklyhood reaches its maximum.
And EM algorithm uses less steps.