Problem Set 8

Jeffrey Kwarsick

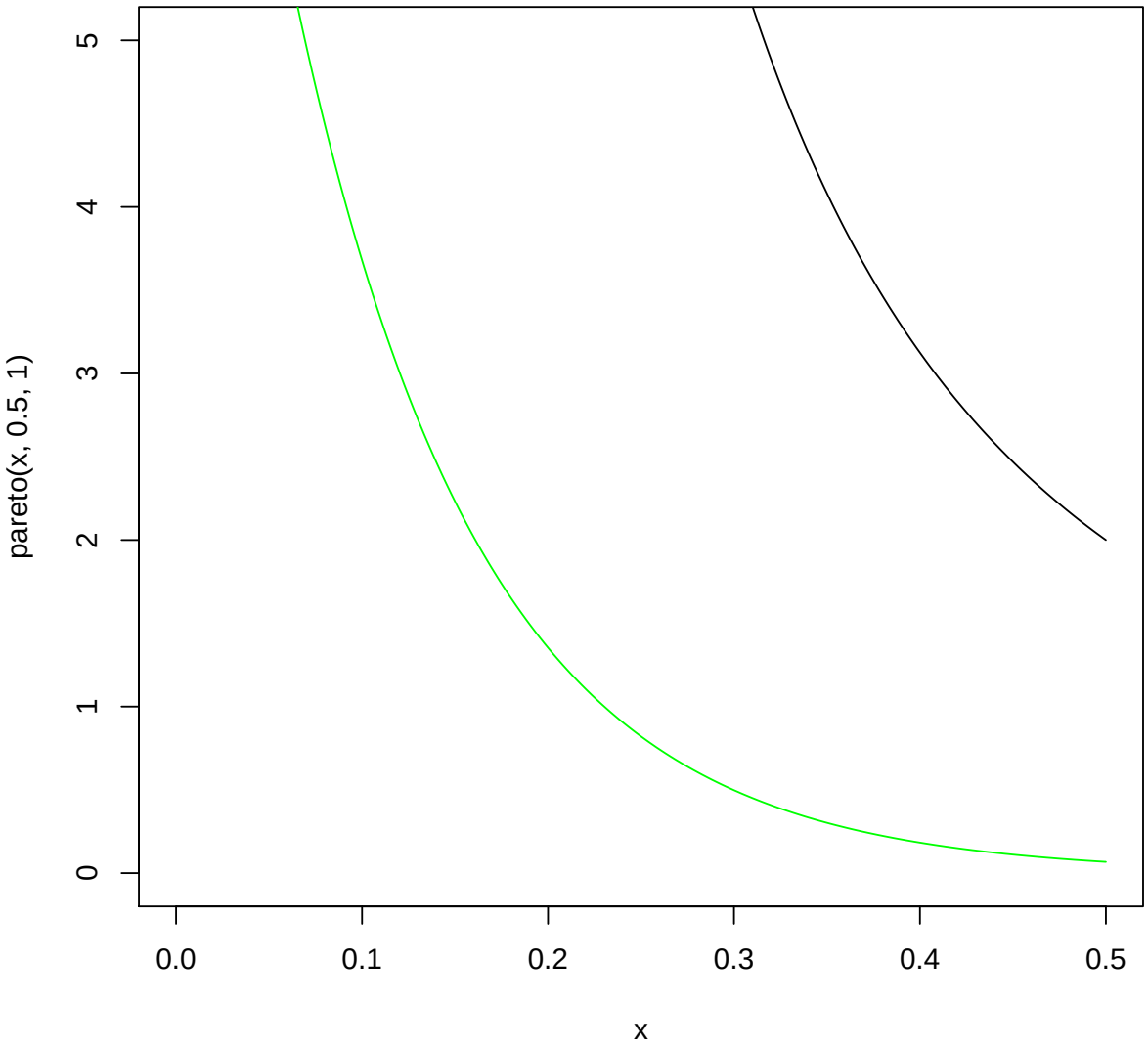
December 1, 2017

1 Problem 1

1.1 Part (a)

The pareto distribution decays more quickly than the exponential distribution. This is illustrated below with a sample plot. The black line is the pareto distribution and green line is the exponential distribution.

```
#################
### Part (a) ###
#################

#pareto pdf
pareto <- function(x, alpha, beta) {
  return( (beta*alpha^beta)/(x^(beta+1)) )
}

#shifted exponential pdf
exp.shft <- function(x, lambda, shft) {
  return( lambda*exp(-lambda*(x-shft)) )
}

#pareto parameters, alpha and beta
alpha  <- 2.0
beta   <- 3.0

#shifted exponential parameter, lambda
lambda <- 1.0
x.shft <- 2.0

#Does the pareto decay faster or more slowly compared to exponential?

#create list of x values to plug into each pdf
#from 0 to 0.5, steps of 0.001
x <- seq(0.0, 0.5, 0.001)

#plotting to show that pareto decays slower compared to exponential
plot(x, pareto(x, 0.5, 1.0), type='l',ylim=c(0,5))
lines(x, exp.shft(x, 10, 0), col='green')
```



1.2 Part (b)

```
#################
### Part (b) ###
#################

#same seed for every iteration of running this problem
set.seed(1)

#inverse cdfs are used for inversion transformation sampling
#used to sample random X_m variable

#pareto pdf
```

```r
pareto <- function(x, alpha, beta) {
  return( (beta*alpha^beta)/(x^(beta+1)) )
}

#inverse cdf of pareto
inverse.cdf.pareto <- function(u, alpha, beta) {
  #where u < 1
  return( alpha*((1 - u)^(-1/beta)) )
}

#shifted exponential pdf, shifted 2 units to right
exp.shft <- function(x, lambda, shft) {
  return( (x >= 2) * lambda * exp( -lambda * (x - shft) ))
}

#inverse cdf of shifted exponential
inverse.cdf.exp.shft <- function(u) {
  #where u < 1
  #lambda = 1
  return( 2 - log(1 - u) )
}

#pareto parameters, alpha and beta
alpha  <- 2.0
beta   <- 3.0

#shifted exponential parameter, lambda
lambda <- 1.0
x.shft <- 2.0

#number of extimators
m <- 10000
#generate m random uniforms for sampling; u < 1
u <- runif(m)

#Sample values from inverse cdf pareto
smpls <- inverse.cdf.pareto(u, alpha = 2, beta = 3)

#f and g functions and weights, w
#f is the hard to sample function (shifted exponential)
#g is the sampling function (pareto)
f <- exp.shft(smpls, lambda = lambda, shft = x.shft)
g <- pareto(smpls, alpha = alpha, beta = beta)

#weights, w
w <- f / g

#X values
x.Ests  <- smpls*w

#X^2 values
x2.Ests <- smpls^2*w
```

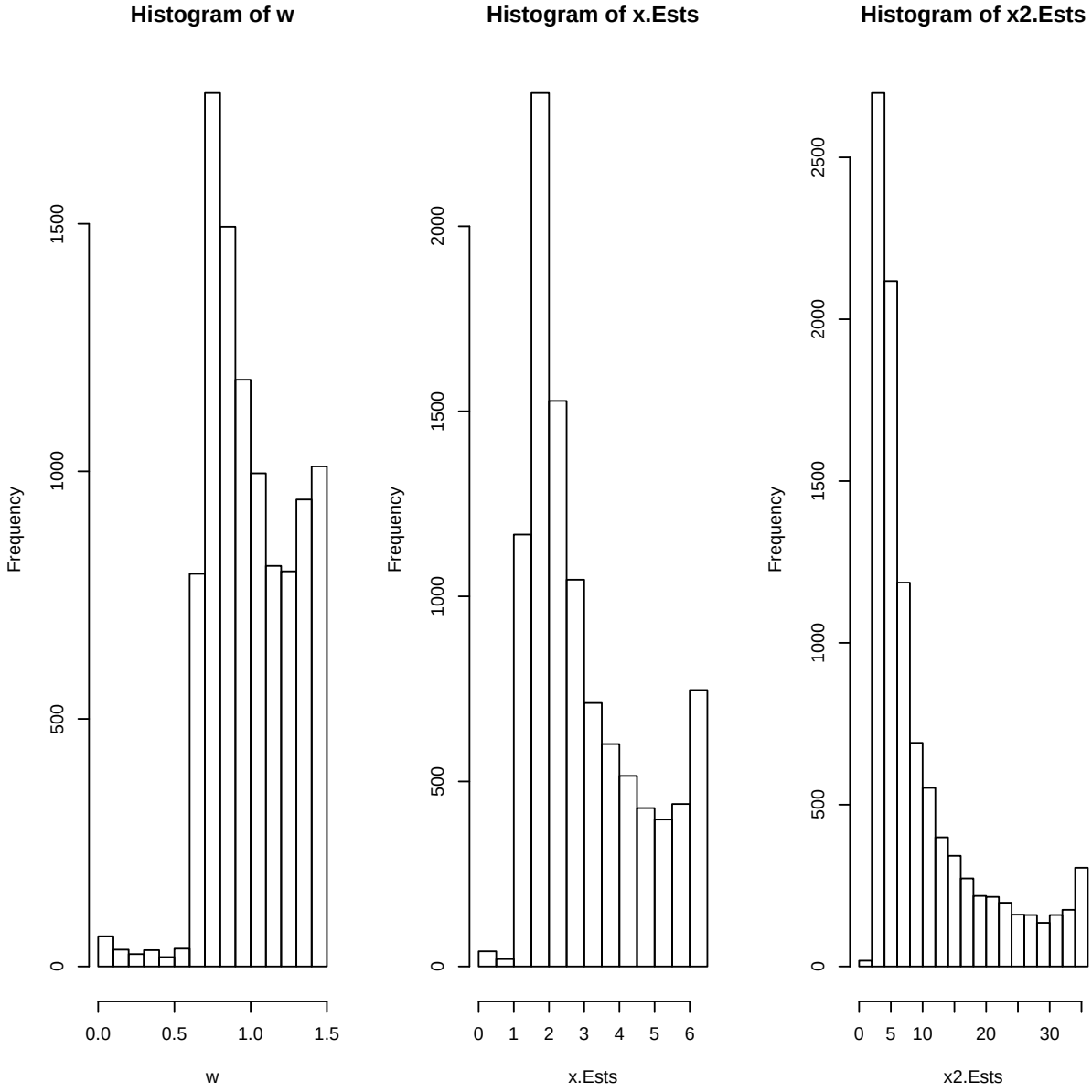
```r
#calculate expectation values for X and X^2
#Estimate of Expectation value of X is
print(mean(x.Ests))
```

```
## [1] 3.012018
```

```r
#Estimate of Expectation value of X^2 is
print(mean(x2.Ests))
```

```
## [1] 10.12722
```

```r
par(mfrow=c(1,3))
#histograms of expectation values for X and X^2
#and for the weights
hist(w)
hist(x.Ests)
hist(x2.Ests)
```

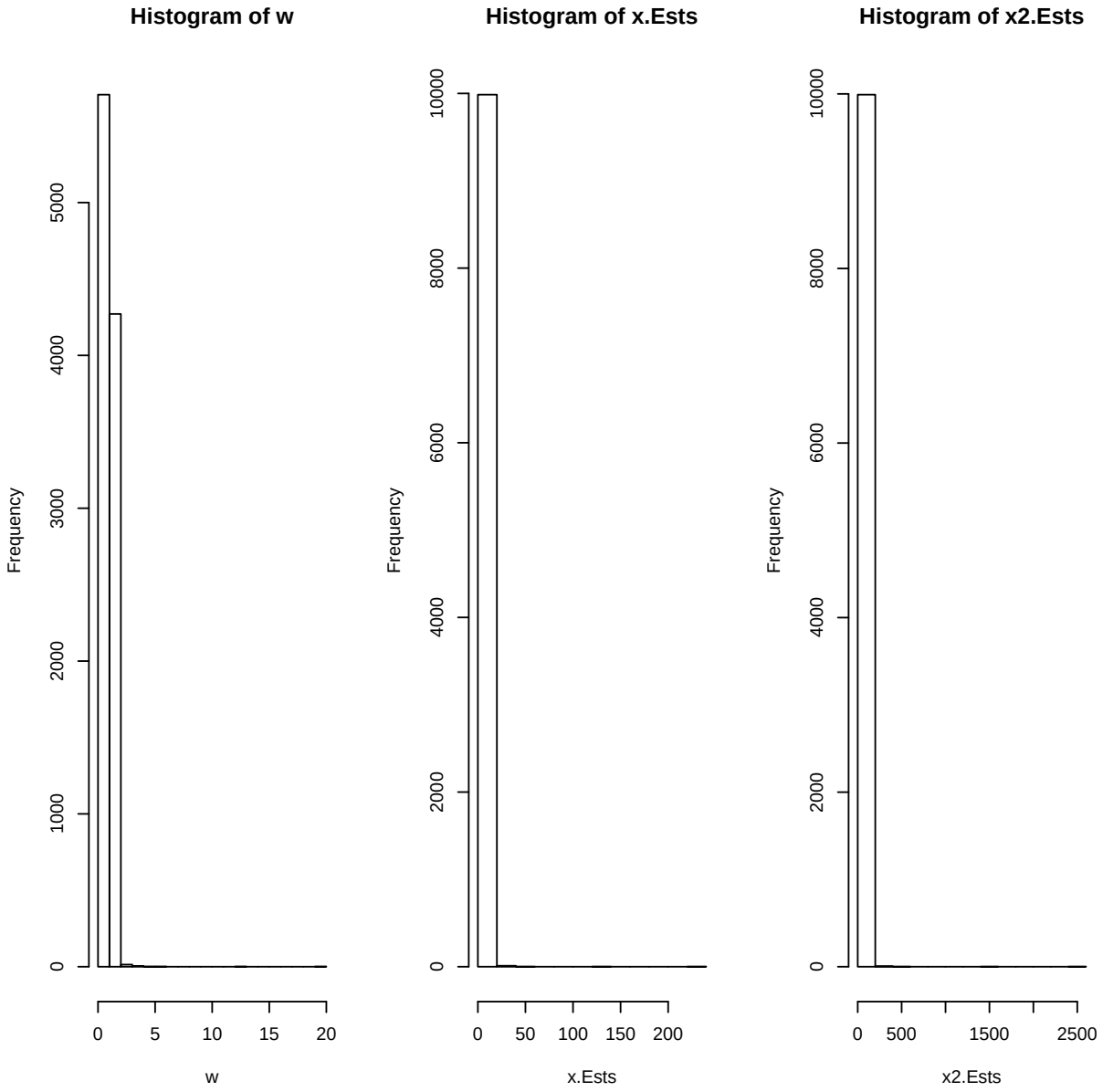


The higher weighted values are sampled with greater frequency compared to the lower weighted values. This is because of the fact that we are using importance sampling.

1.3 Part (c)

```
#################
### Part (c) ###
#################

#Proceeds in the same manner as Part (b)
#only the functions are switched
set.seed(1)
m  <- 10000
u <- runif(m)
```

```r
smpls <- inverse.cdf.exp.shft(u)

f <- pareto(smpls, alpha = alpha, beta = beta)
g <- exp.shft(smpls, lambda = lambda, shft = x.shft)

w <- f / g

x.Ests  <- smpls*w
x2.Ests <- smpls^2*w

print("Estimate for Expectation Value of X is ")

## [1] "Estimate for Expectation Value of X is "

print(mean(x.Ests))

## [1] 2.903723

print("Estimate for Expectation Value of X^2 is ")

## [1] "Estimate for Expectation Value of X^2 is "

print(mean(x2.Ests))

## [1] 9.852741

par(mfrow=c(1,3))
#histograms of weights, and values for expectation values of X and X^2
hist(w)
hist(x.Ests)
hist(x2.Ests)
```

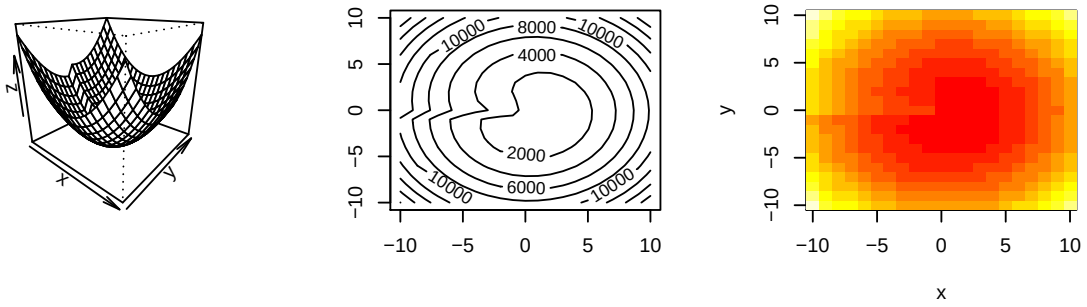


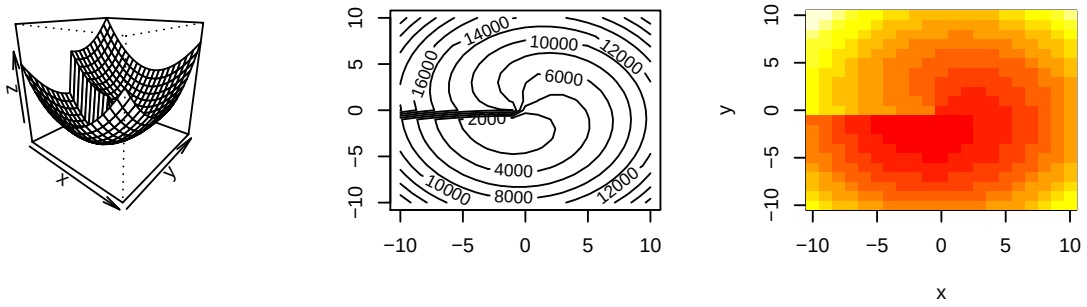
The weights in this version of the importance sampling is not as extreme as compared to the importance sampling of the previous part.

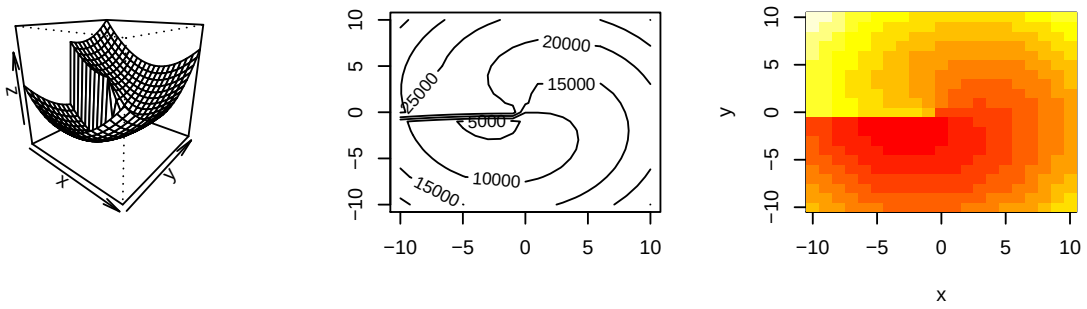
2 Problem 2

```
###################
### Question 2 ###
###################

### Taken from ps8.R provided by Chris ###
theta <- function(x1,x2) atan2(x2, x1)/(2*pi)

#helical function
```

```r
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)
}
### Taken from ps8.R provided by Chris ###

par(mfrow=c(3,3))
#Create perspective, contour, and image plots with constant z-conditions
#scan over the values of x and y
#[-100, 100] with step size = 1
#hold z = -1, -5 and -10
for (k in c(1,5,10)) {
  x=seq(-10,10,1)
  y=seq(-10,10,1)
  x.len= length(x)
  y.len = length(y)
  z=array(0,dim=c(x.len,y.len))
  #scan over all values of x and y and hold z constant
  for (i in 1:x.len) for (j in 1:y.len) z[i,j] = f(c(x[i],y[j],-k))
    #create plots -- perspective, contour, and image
    persp(x, y, z, phi = 20, theta = 40)
    contour(x, y, z)
    image(x, y, z)
}
```








```
#Compare optimization methods for all values in x,y, and z ranges
#[-100,100], step-size = 1
x.seq <- seq(-10,10,1)
y.seq <- seq(-10,10,1)
z.seq <- seq(-10,10,1)

#create empy vectors for optimization outputs
optim.output <- c()
nlm.output   <- c()

#conducting optimization, scanning over all values
#both optim and nlm studied
for (i in x.seq) {
  for (j in y.seq) {
```

```
    for (k in z.seq) {
      tmp.optim <- optim(c(i,j,k), fn = f)
      tmp.nlm   <- nlm(p = c(i,j,k), f = f)
      #save outputs to initialized lists
      optim.output <-  rbind(optim.output, c(tmp.optim$par, tmp.optim$value))
      nlm.output   <-  rbind(nlm.output, c(tmp.nlm$estimate, tmp.nlm$minimum))
    }
  }
}
#Print the resulting outputs from each optimization
head(optim.output)
```

```
##           [,1]          [,2]          [,3]         [,4]
## [1,] 1.0000192  0.0017768600  0.0035919425 7.132079e-05
## [2,] 0.9986558 -0.0008981530 -0.0018195403 1.989508e-04
## [3,] 1.0000462 -0.0063633215 -0.0099802284 1.021983e-04
## [4,] 0.9995174  0.0064526016  0.0096817711 1.501879e-04
## [5,] 1.0002442 -0.0014208359 -0.0027643164 3.900827e-05
## [6,] 0.9999825 -0.0004433395 -0.0008347427 2.394584e-06
```

```
head(nlm.output)
```

```
##           [,1]          [,2]          [,3]         [,4]
## [1,] 1.0000000 -2.788850e-10 -4.093529e-10 4.726858e-19
## [2,] 1.0000000 -8.894507e-09 -7.034610e-09 5.131968e-15
## [3,] 1.0000000  1.858146e-09 -2.066969e-09 3.142904e-15
## [4,] 0.9999995 -8.223398e-05 -1.300839e-04 1.701008e-08
## [5,] 1.0000000 -5.687298e-11  1.608843e-10 1.285024e-17
## [6,] 1.0000000  1.206080e-09  1.864384e-09 1.203165e-17
```

```
#signifcant use of decimals make all answers unique
#rounding to reduce this issue
#rounding to two decimal places
#printing unique results from rounded lists

head(unique(round(optim.output, 2)))
```

```
##      [,1]  [,2]  [,3] [,4]
## [1,]    1  0.00  0.00    0
## [2,]    1 -0.01 -0.01    0
## [3,]    1  0.01  0.01    0
## [4,]    1  0.00  0.01    0
## [5,]    1  0.02  0.02    0
## [6,]    1  0.00 -0.01    0
```

```
head(unique(round(nlm.output, 2)))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]   -8    0    5 4925
## [3,]   -7    0    5 3625
## [4,]   -6    0    5 2525
## [5,]   -5    0    5 1625
## [6,]   -5    0    6 1736
```

3 Problem 3

3.1 Part (a)

Mathematical derivation was done by hand and is attached to the hard-copy of this problem set. Below is the pseudo-implementation of the EM algorithm for a censored linear regression. In order to complete the math, I referenced the following article –

Park, Chanseok, Seong Beom, Lee. (2003). *Parameter Estimation from Censored Samples using Expectation-Maximumization Algorithm.* Clemson University. https://arxiv.org/pdf/1203.3880.pdf.

```
##################
### Question 3 ###
##################

#cen.reg.EM <- name of EM algorithm
#theta <- (b0, b1, s2) wher s2 = sigma^2
#epsilon <- convergence condition tolerance
#X <- xs needed for generation of slopes
#Y <- Y_observed values
#Z <- censored data values
#tau <- censoring condition


cen.reg.EM <- function(theta, epsilon, X, Y, Z, tau) {
  t <- 0 #iterations counter
  converge.cond <- FALSE #convergence initialization

  while(!converge.cond) {
    #conduct calculations of theta(t+1)
    #obtained by taking the derivatives of the
    #log-likelihood of theta
    prev.theta <- theta
    #tau star of truncated normal
    tau1 <- (tau - (prev.theta[2]+prev.theta[1]*X)/sqrt(prev.theta[3]))
    #length of data
    n <- length(X)
    #position of truncation
    c <- length(Z)
    #rho(tau1): rho of tau star (truncated normal)
    rho <- dnorm(tau1) / (1 - pnorm(tau1))

    beta0.update <- n^-1*(sum(Z) + (n - c)*(prev.theta[1] + prev.theta[2]*X
                                      + sum(prev.theta[3]*rho)))
    beta1.update <- n^-1*(sum(Z) + (n - c)*(prev.theta[1] + theta[2]*X
                                      + sum(prev.theta[3]*rho)))
    sigma.update <- n^-1*(sum(Z^2) + (n - c)*(prev.theta[1] + prev.theta[2]*X)
                          *((prev.theta[1] + prev.theta[2]*X) + prev.theta[3]) +
                            sum((prev.theta[1] + prev.theta[2]*X + tau)*sqrt(prev.theta[3])*rho))
                          + (1/n^2)*(sum(Z) + (n - c)*(prev.theta[1] + prev.theta[2]*X
                          + sum(prev.theta[3]*rho)))
    #save to new theta
    theta <- c(beta0.update, beta1.update, sigma.update)
    t <- t + 1 #iteration step
```

```
    #Exit condition for the algorithm
    if(max(abs(theta - prev.theta)) < epsilon ) {
      converge.cond <- TRUE
    }
  }
  #return converged values and number of iterations
  return(c(theta, t))

}
```

3.2 Part (b)

For initialization conditions, I propose that the intercept, β_0, is set to zero, the slope, β_1, is set to 1, and the variance, σ^2, is set to 0.5. This way, all of the values are on the same scale.

3.3 Part (c)

3.4 Part (d)