Stat 243 PS 8

Riv Jenkins

1 a)

Pareto decays more slowly than an exponential distribution because e^{-x} decays more quickly than $x^{\beta+1}$.

b)

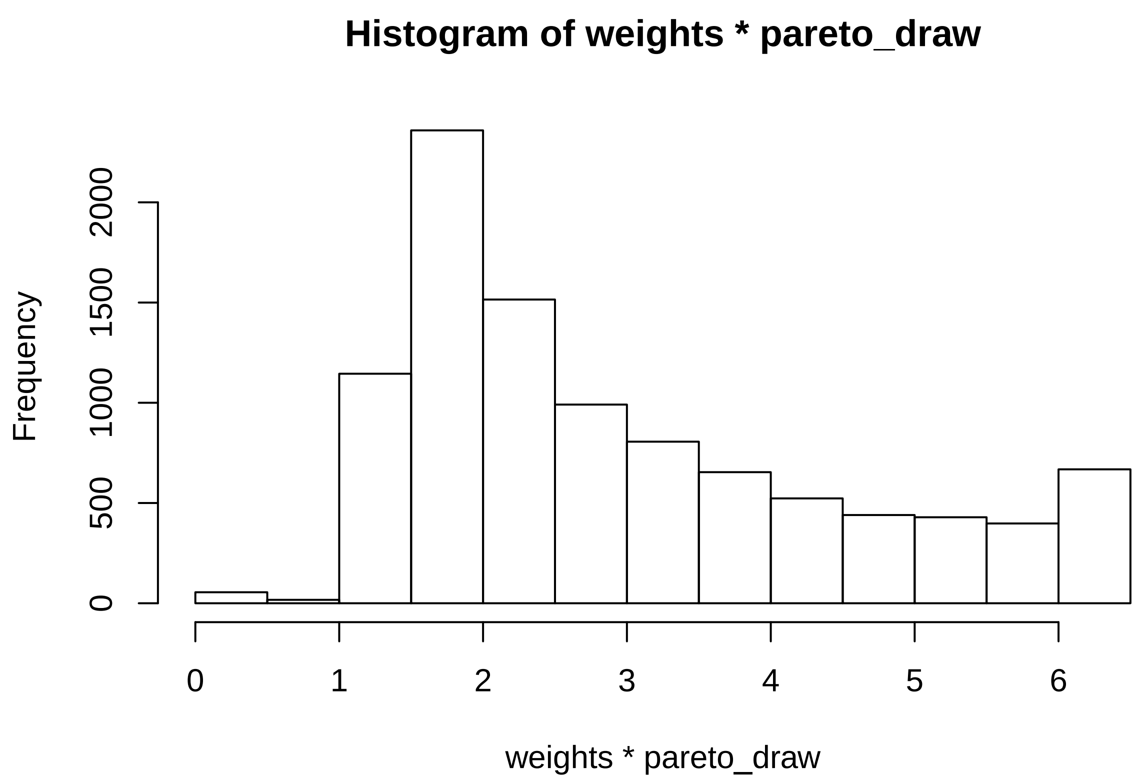
```
m=10000
pareto_draw = rpareto(m, 2, 3)
weights = dexp(pareto_draw-2)/dpareto(pareto_draw, 2, 3)
E_X = 1/m * sum(weights*pareto_draw)
E_X2 = 1/m * sum(weights*pareto_draw**2)
print(paste0("E(X) estimate: ", E_X))
```

```
## [1] "E(X) estimate: 2.99242066538142"
```

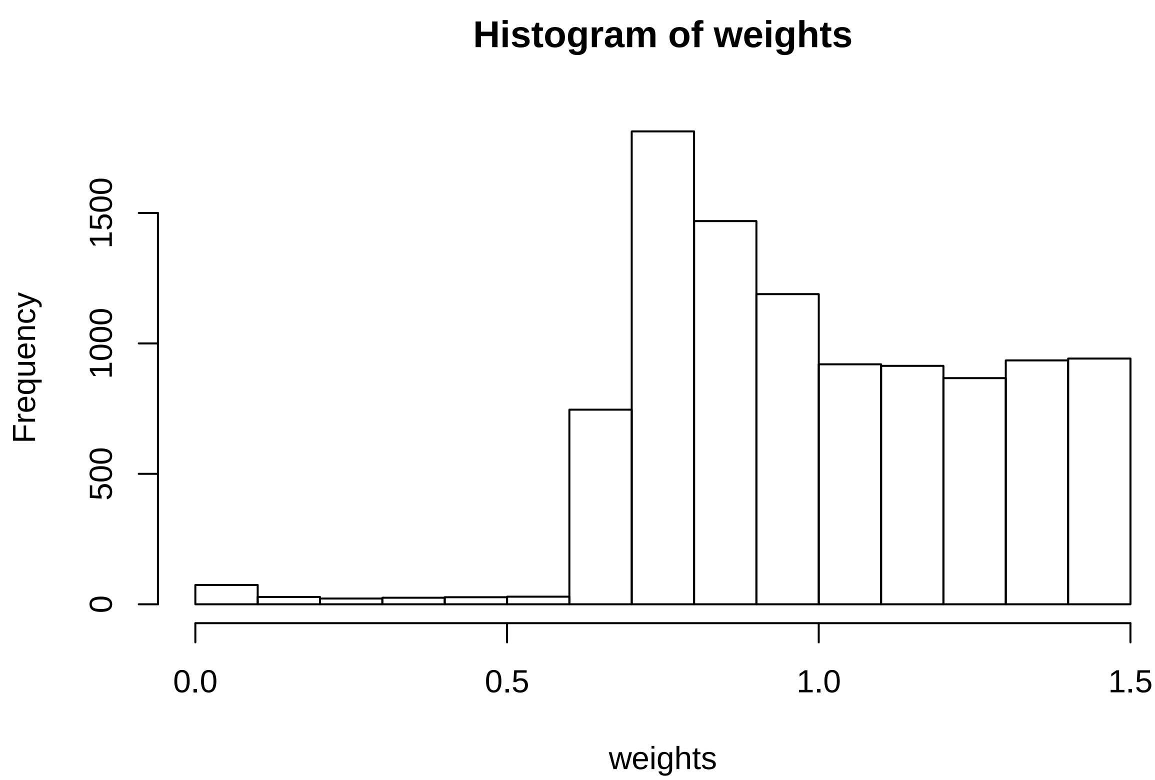
```
print(paste0("E(X^2) estimate: ", E_X2))
```

```
## [1] "E(X^2) estimate: 9.95293005060605"
```

```
hist(weights*pareto_draw)
```



```r
hist(weights)
```



There are some larger weights from this sampling method, there are not any large outliers which would indicate an unnaturally large variance.

c)

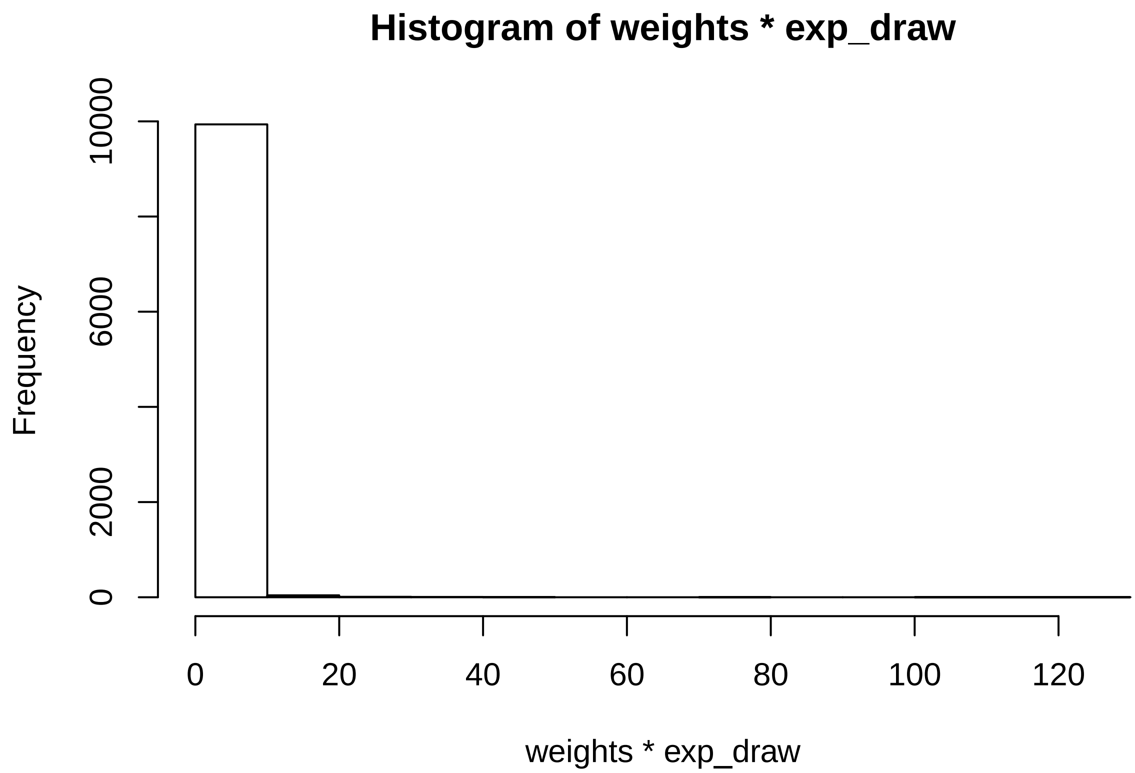
```r
m=10000
exp_draw = rexp(m)+2
weights = dpareto(exp_draw, 2, 3)/dexp(exp_draw-2)
E_X = 1/m * sum(weights*exp_draw)
E_X2 = 1/m * sum(weights*exp_draw**2)
print(paste0("E(X) estimate: ", E_X))
```

```
## [1] "E(X) estimate: 2.9049593889998"
```

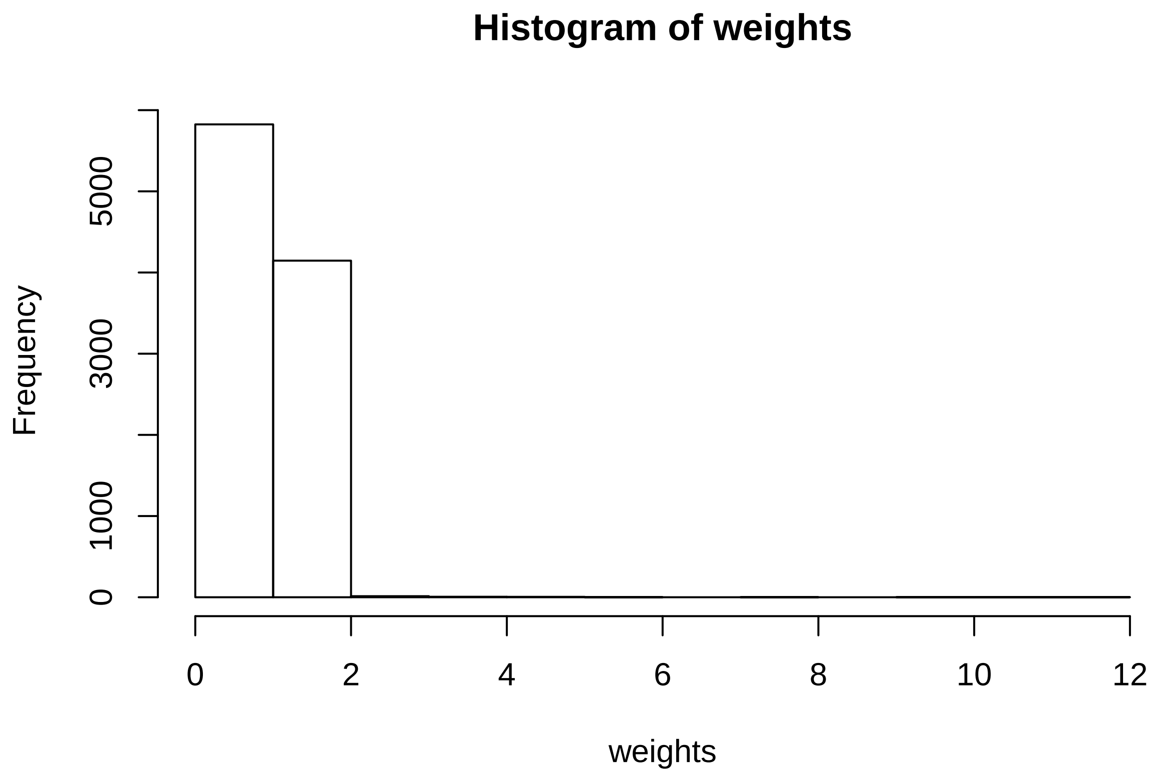
```r
print(paste0("E(X^2) estimate: ", E_X2))
```

```
## [1] "E(X^2) estimate: 9.91350591708933"
```

```r
hist(weights*exp_draw)
```




```r
hist(weights)
```

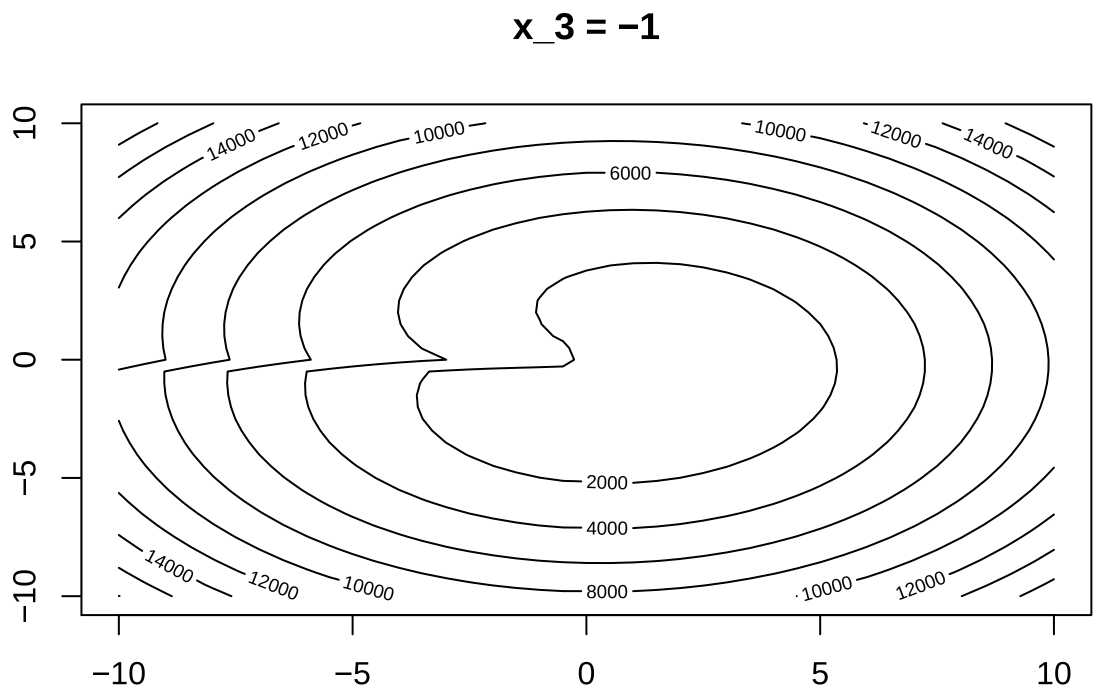


With this sampling method we can see large outlier weights which will skew the results for our estimates. This method is not as good as that in part b.

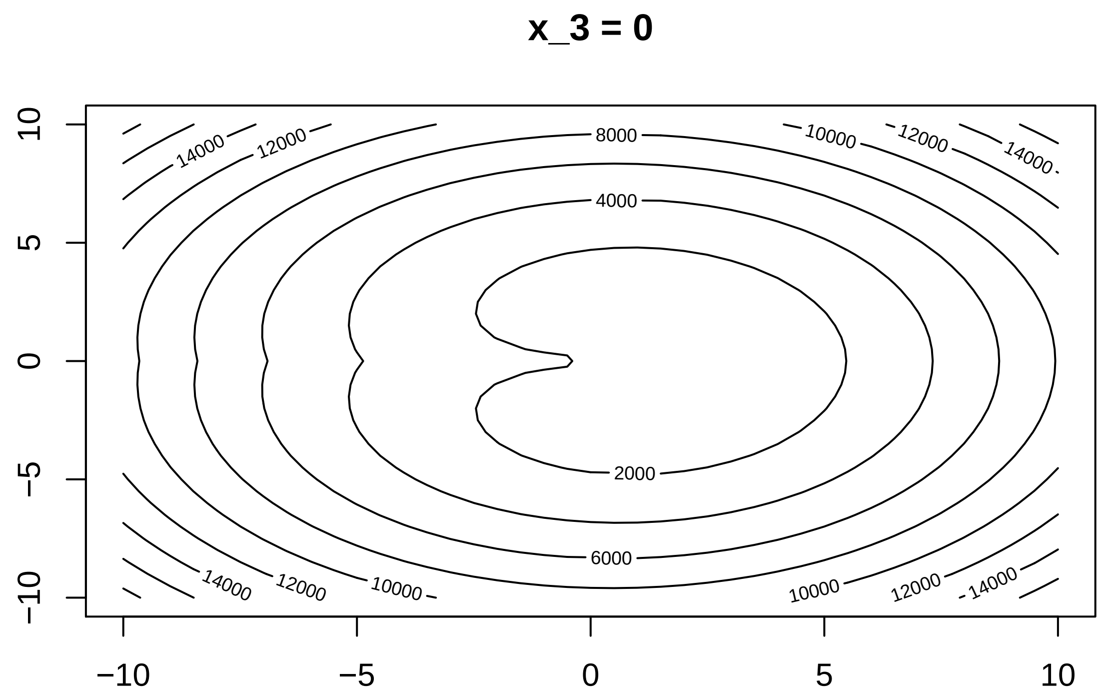
2

```
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)

}
x_1 = seq(-10, 10, 0.5)
x_2 = seq(-10, 10, 0.5)
x_3 = vector(length = length(x_1))-1
z = matrix(nrow = length(x_1), ncol = length(x_1))
for (i in 1:length(x_1)){
  for (j in 1:length(x_2)){
    z[i,j] = f(c(x_1[i], x_2[j], x_3[i]))
  }
}
```

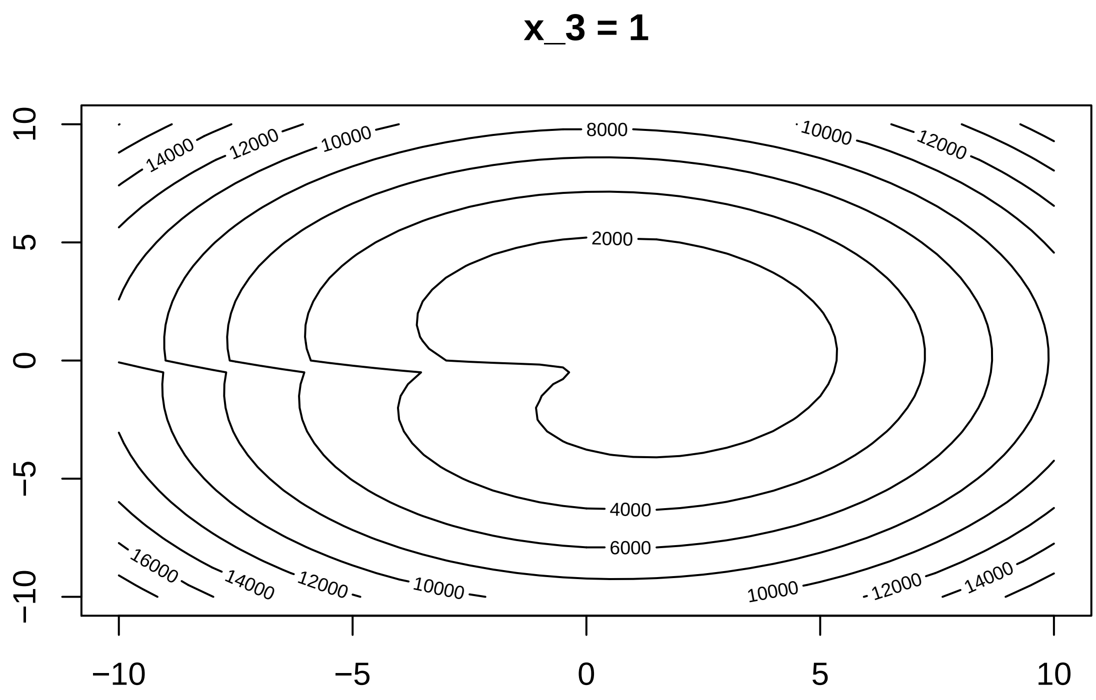
```
contour(x_1, x_2, z)
title("x_3 = -1")
```




```
x_1 = seq(-10, 10, 0.5)
x_2 = seq(-10, 10, 0.5)
x_3 = vector(length = length(x_1))
z = matrix(nrow = length(x_1), ncol = length(x_1))
for (i in 1:length(x_1)){
  for (j in 1:length(x_2)){
    z[i,j] = f(c(x_1[i], x_2[j], x_3[i]))
  }
}
contour(x_1, x_2, z)
title("x_3 = 0")
```




```
x_1 = seq(-10, 10, 0.5)
x_2 = seq(-10, 10, 0.5)
x_3 = vector(length = length(x_1))+1
z = matrix(nrow = length(x_1), ncol = length(x_1))
for (i in 1:length(x_1)){
  for (j in 1:length(x_2)){
    z[i,j] = f(c(x_1[i], x_2[j], x_3[i]))
  }
}
contour(x_1, x_2, z)
title("x_3 = 1")
```




```r
optim(c(3, 0, 0), f)
```

```
## $par
## [1] 1.0000560801 0.0006847037 0.0011448265
##
## $value
## [1] 1.931891e-06
##
## $counts
## function gradient
##      180       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
nlm(f, c(3, 0, 0))
```

```
## $minimum
## [1] 2.739949e-12
##
## $estimate
## [1]  1.000000e+00 -1.044732e-06 -1.641194e-06
##
## $gradient
```

```
## [1] -2.818217e-09 -6.858795e-06  1.027122e-06
##
## $code
## [1] 2
##
## $iterations
## [1] 4
```

```
optim(c(1, 0, 0), f)
```

```
## $par
## [1] 1 0 0
##
## $value
## [1] 0
##
## $counts
## function gradient
##      158       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The minimum value seems to be at (1, 0, 0) with a function value of 0. Both the optim and nlm solvers seem to agree on this point.

3 a)

We first have to set up the likelihood function.

$$L(Y, Z|\theta) = \prod_{i=1}^{m} \phi(y_i) \prod_{i=m+1}^{n} \phi(z_i)$$

where $n - m = c$. We then take the log.

$$\log(L) = \sum_{i=1}^{m} \left[-\log(\sigma) - \frac{(y_i - \mu)^2}{2\sigma^2} \right] + \sum_{i=m+1}^{n} \left[-\log(\sigma) - \frac{(z_i - \mu)^2}{2\sigma^2} \right]$$

where $\mu = \beta_0 + \beta_1 x_i$. Now we must take the expectation with respect to Z given $z_i > \tau, \mu = \mu_t, \sigma = \sigma_t$.

$$E[\log(L)] = \sum_{i=1}^{m} \left[-\log(\sigma) - \frac{(y_i - \mu)^2}{2\sigma^2} \right] + \sum_{i=m+1}^{n} \left[-\log(\sigma) - \frac{1}{2\sigma^2} E[(z_i - \mu)^2] \right]$$

Now, we expand out $E[(z_i - \mu)^2]$.

$$E[(z_i - \mu)^2] = E[z_i^2] - 2\mu E[z_i] + \mu^2$$

We then use the properties of expectation of a truncated normal distribution to expand out these expectations.

$$E[(z_i - \mu)^2] = \sigma_t^2 + \sigma_t^2 \tau_t^* \rho_t(\tau^*) - \sigma_t^2 \rho_t(\tau^*)^2 + \mu_t^2 + 2\mu_t \sigma_t \rho_t(\tau^*) + \sigma_t^2 \rho_t(\tau^*)^2 - 2\mu(\mu_t + \sigma_t \rho_t(\tau^*)) + \mu^2$$

$$E[(z_i - \mu)^2] = \sigma_t^2 + \sigma_t^2 \tau_t^* \rho_t(\tau^*) - \sigma_t^2 \rho_t(\tau^*)^2 + \sigma_t^2 \rho_t(\tau^*)^2 + \mu_t^2 + 2\mu_t \sigma_t \rho_t(\tau^*) - 2\mu(\mu_t + \sigma_t \rho_t(\tau^*)) + \mu^2$$

$$E[(z_i-\mu)^2] = \sigma_t^2 + \sigma_t^2\tau_t^*\rho_t(\tau^*) - \sigma_t^2\rho_t(\tau^*)^2 + (\mu_t + \sigma_t\rho_t(\tau^*))^2 - 2\mu(\mu_t + \sigma_t\rho_t(\tau^*)) + \mu^2$$
$$E[(z_i-\mu)^2] = \sigma_t^2 + \sigma_t^2\tau_t^*\rho_t(\tau^*) - \sigma_t^2\rho_t(\tau^*)^2 + ((\mu_t + \sigma_t\rho_t(\tau^*)) - \mu)^2$$
$$E[(z_i-\mu)^2] = \sigma_t^2 + \sigma_t^2\tau_t^*\rho_t(\tau^*) - \sigma_t^2\rho_t(\tau^*)^2 + (E[z_i] - \mu)^2$$

Now we can plug this back into the log likelihood function.

$$E[\log(L)] = \sum_{i=1}^{m}\left[-\log(\sigma) - \frac{(y_i-\mu)^2}{2\sigma^2}\right] + \sum_{i=m+1}^{n}\left[-\log(\sigma) - \frac{1}{2\sigma^2}(\sigma_t^2 + \sigma_t^2\tau_t^*\rho_t(\tau^*) - \sigma_t^2\rho_t(\tau^*)^2 + (E[z_i]-\mu)^2)\right]$$

We then need to maximize this in terms of β_0, β_1, and σ. It is clear that the best values of β_0 and $beta_1$ can be found by running a least squares regression on the data using $E[z_i]$ for the values with $z_i > \tau$. We can then use these values to maximize in terms of σ by taking the gradient and setting it equal to 0. For simplicity let $K_i = (\sigma_t^2 + \sigma_t^2\tau_t^*\rho_t(\tau^*) - \sigma_t^2\rho_t(\tau^*)^2 + (E[z_i]-\mu)^2)$. Now we take the derivative with respect to σ and set it equal to 0.

$$0 = \sum_{i=1}^{m}\left[-\frac{1}{\sigma} + \frac{(y_i-\mu)^2}{\sigma^3}\right] + \sum_{i=m+1}^{n}\left[-\frac{1}{\sigma} + \frac{1}{\sigma^3}K_i\right]$$
$$n = \sum_{i=1}^{m}\frac{(y_i-\mu)^2}{\sigma^2} + \sum_{i=m+1}^{n}\frac{1}{\sigma^2}K_i$$
$$\sigma^2 = \frac{\sum_{i=1}^{m}(y_i-\mu)^2 + \sum_{i=m+1}^{n}K_i}{n}$$

b)

$\beta_0^{(0)}$ and $\beta_1^{(0)}$ can be estimated using the uncensored data and fitting a regression to it. An easily calculated σ_0 would be standard deviation of the error from said regression (again using only the data below the censor threshold).

c)

```r
#generate the data

set.seed(1)

n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

x <- runif(n)

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

## parameters chose such that signal in data is moderately strong

## estimate divided by std error is ~ 3
tau = 4
Censored = yComplete > 4
UnCensored = yComplete <= 4
print(paste0(sum(Censored), " percent is censored"))
```

```
## [1] "20 percent is censored"
```

```
CalcSigma <- function(yvals, x_uncen, new_beta0, new_beta1, Ez, x_cen, old_beta0,
                      old_beta1, old_sigma){
  #first calculate K_i
  tau_star = vector(length = length(x_cen))
  rho = vector(length = length(x_cen))
  K = vector(length = length(x_cen))
  for (i in 1:length(x_cen)){
    tau_star[i] = (tau - (old_beta0 + old_beta1*x_cen[i]))/old_sigma
    rho[i] = pnorm(tau_star[i], mean = old_beta0 + old_beta1*x_cen[i])/(1-dnorm(tau_star[i], mean = old_
    K[i] = old_sigma**2 * (1 + tau_star[i] * rho[i] - rho[i]**2)
  }
  K = K + (Ez - (new_beta0 + new_beta1*x_cen))**2

  #calculate Yerr
  Yerr = (yvals - (new_beta0 + new_beta1*x_uncen))**2

  sigma2 = (sum(Yerr) + sum(K)) / (length(x_uncen) + length(x_cen))

  return(sqrt(sigma2))
}
```

```
#inital parameters
old_reg = lm(yComplete[UnCensored]~x[UnCensored])
old_beta0 = old_reg$coefficients[1]
old_beta1 = old_reg$coefficients[2]
old_sigma = sd(old_reg$residuals)

print("Beta_0 Beta_1 Sigma")
```

```
## [1] "Beta_0 Beta_1 Sigma"
```

```
for (i in 1:10){

  #calculate expected value of z_i
  tau_star = vector(length = length(x[Censored]))
  rho = vector(length = length(x[Censored]))
  for (i in 1:length(x[Censored])){
    tau_star[i] = (tau - (old_beta0 + old_beta1*x[Censored][i]))/old_sigma
    rho[i] = pnorm(tau_star[i], mean = old_beta0 + old_beta1*x[Censored][i])/(1-dnorm(tau_star[i], mean
  }
  Ez = old_beta0 + old_beta1*x[Censored] + old_sigma*rho

  #calculate new mu (beta_0 and beta_1)
  new_x = append(x[Censored], x[UnCensored])
  Y_est = append(Ez, yComplete[UnCensored])
  new_reg = lm(Y_est~new_x)
  new_beta0 = new_reg$coefficients[1]
  new_beta1 = new_reg$coefficients[2]

  #calculate new sigma
  new_sigma = CalcSigma(yComplete[UnCensored], x[UnCensored], new_beta0, new_beta1,
                        Ez, x[Censored], old_beta0, old_beta1, old_sigma)
```

```
  #reset parameters
  old_sigma = new_sigma
  old_beta0 = new_beta0
  old_beta1 = new_beta1
  print(paste0(round(old_beta0, digits=5)," ", round(old_beta1, digits=5)," ", round(old_sigma, digits=5
}
```

```
## [1] "0.3549 2.03043 1.79409"
## [1] "0.44379 1.8634 1.81736"
## [1] "0.45542 1.84646 1.82603"
## [1] "0.45774 1.84254 1.82806"
## [1] "0.45824 1.84172 1.82856"
## [1] "0.45836 1.84152 1.82869"
## [1] "0.45839 1.84147 1.82872"
## [1] "0.45839 1.84146 1.82872"
## [1] "0.45839 1.84146 1.82873"
## [1] "0.4584 1.84146 1.82873"
```

These numbers do not seem right, but I could not find the error in my code. I believe the error is in the code not in the derivation.