Problem Set 8

Cameron Adams

December 1, 2017

1 Lets consider importance sampling and explore ...

1.1 Does the tail of the Pareto decay more quickly or more slowly than that of an exponential distribution?

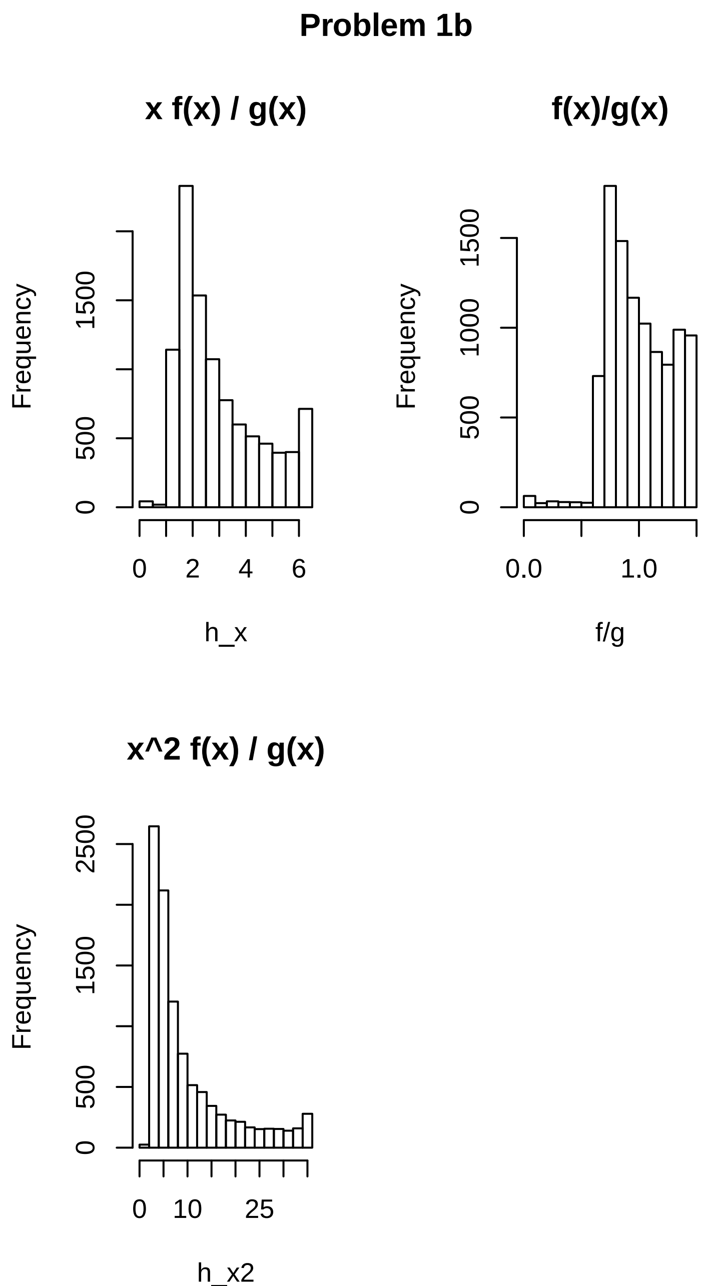
The pareto distribution decays more slowly than the exponential distribution

1.2 Suppose f is an exponential density with parameter value ...

```
rm(list=ls())

require(extraDistr)

# parms
m <- 10000
a <- 3
b <- 2

#generate x according to parato
x <- rpareto(m, a = a, b = b)

#generate f(x)
f <- ifelse(x < 2, 0, dexp(x - 2))

#generate g(x)
g <- dpareto(x, a = a, b = b)

#check g(x) satisfies paraeto conditions
sum(g > 2 & g < 1e9)

## [1] 0

#h(x) = h * f / g
h_x <- x*f/g # x
h_x2 <- x^2*f/g # x^2

#get expection
mean(h_x)

## [1] 3.002851

mean(h_x2)
```

```
## [1] 9.998935
```

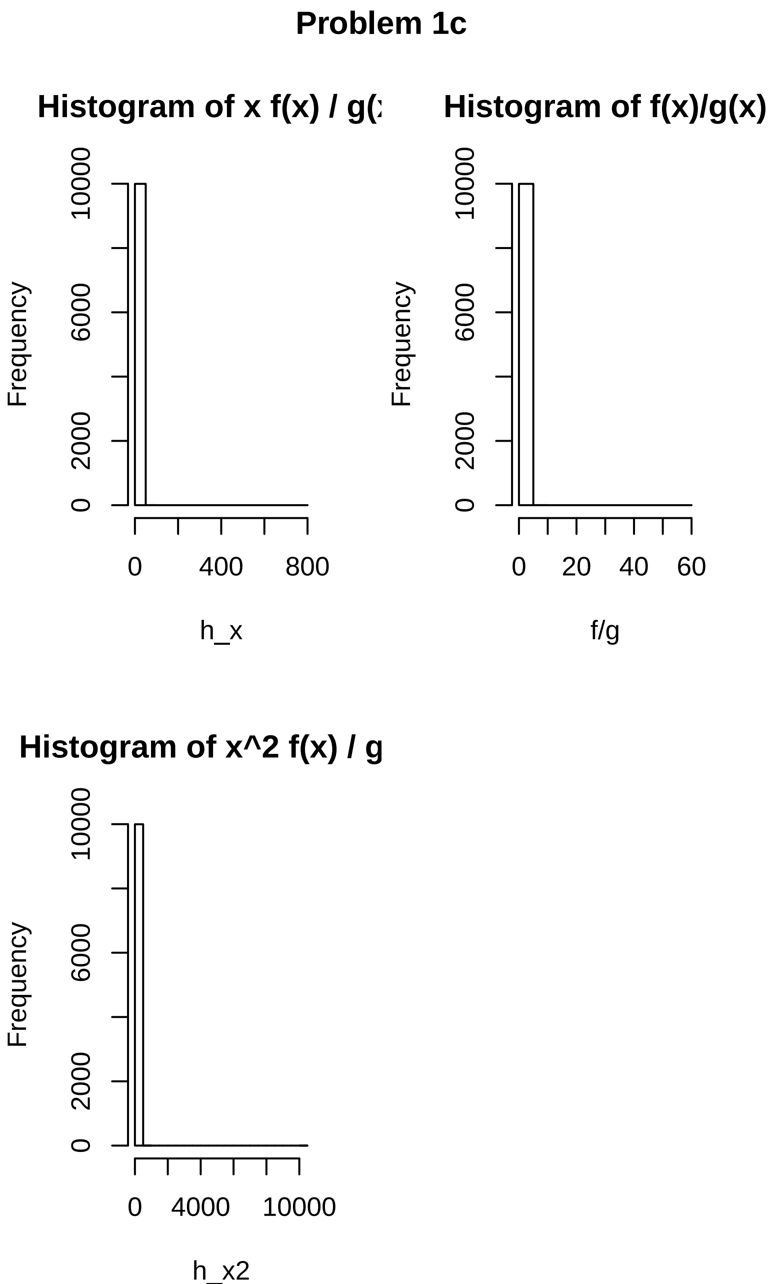
```r
#histograms
par_default <- par(no.readonly = TRUE)
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
hist(h_x, main = "x f(x) / g(x)")
hist(f / g, main = "f(x)/g(x)")
hist(h_x2, main = "x^2 f(x) / g(x)")
mtext(outer = T, text = "Problem 1b", font = 2)
par(par_default)
```



It appears that large x values have very small weights and x values approaching 2 will have large weights

1.3 Now suppose f is the Pareto distribution described above and our sampling...

```r
#generate x and f
x <- rexp(m)+2 #exp
f <- dpareto(x, a, b) #pareto

#generate g
g <- ifelse(x<2, 0, dexp(x-2))
h_x <- x*f/g
h_x2 <- x^2*f/g

#get expection
mean(h_x)

## [1] 2.950998

mean(h_x2)

## [1] 10.56279

#histograms
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
hist(h_x, main = "Histogram of x f(x) / g(x)")
hist(f / g, main = "Histogram of f(x)/g(x)")
hist(h_x2, main = "Histogram of x^2 f(x) / g(x)")
mtext(outer = T, text = "Problem 1c", font = 2)
par(par_default)
```



The version of smapling will have large weights for values close 0, and smaller weights to values close to 0, with no/zero weight with values ¿ 2.

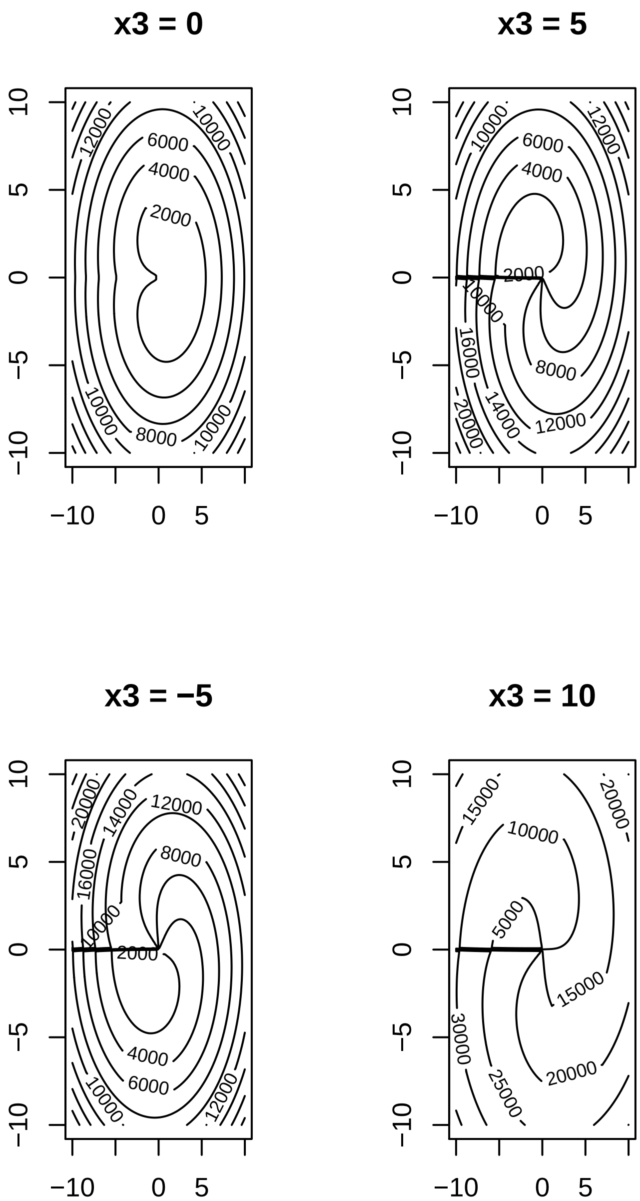
2 Consider the helical valley function ...

```
source()
```

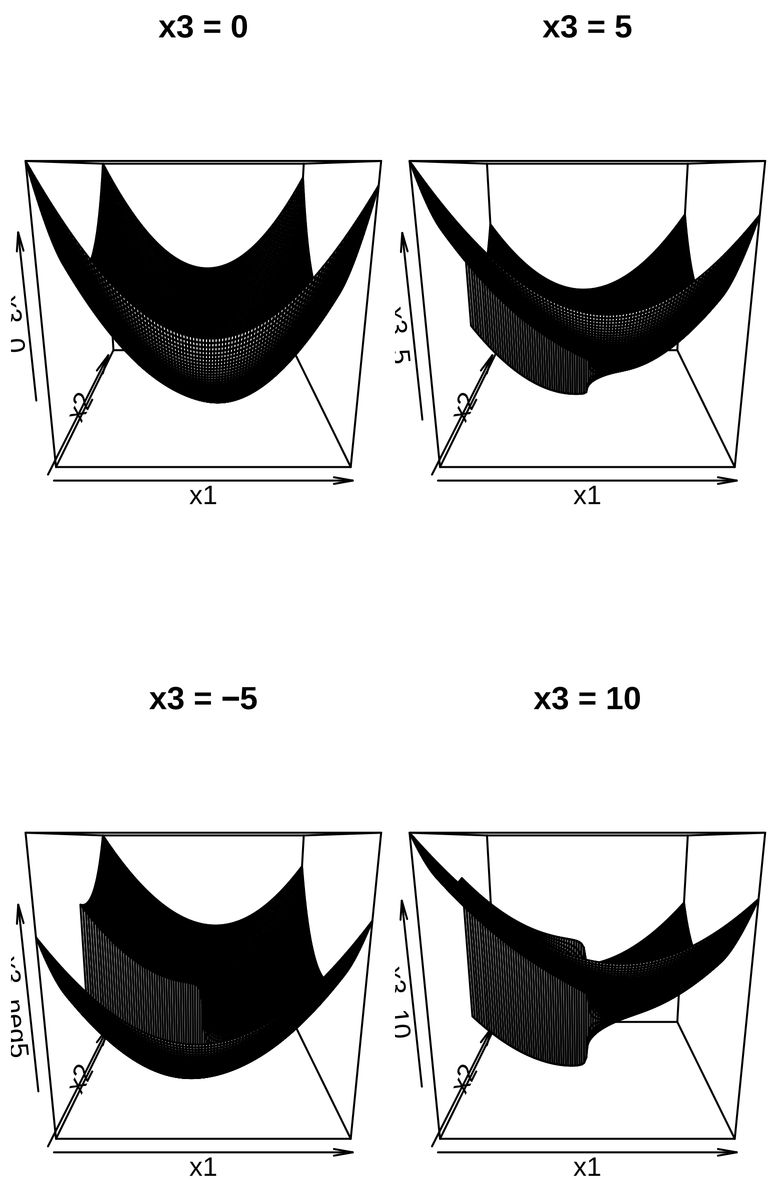
```
## Error in source():  argument "file" is missing, with no default
```

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

```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)
}

source("./ps8.R")

#generate x1 and x2
x1 <- x2 <- seq(-10, 10, length.out = n)
x <- cbind(expand.grid(x1, x2), 0)

#x3 = 0
x <- cbind(expand.grid(x1, x2), 0)
x3_0    <- matrix(apply(cbind(expand.grid(x1, x2), 0), 1, f), ncol = n)
x3_5    <- matrix(apply(cbind(expand.grid(x1, x2), 5), 1, f), ncol = n)
x3_neg5 <- matrix(apply(cbind(expand.grid(x1, x2), -5), 1, f), ncol = n)
x3_10   <- matrix(apply(cbind(expand.grid(x1, x2), 10), 1, f), ncol = n)

#plot contours
par(mfrow = c(2, 2))
contour(x1, x2, x3_0, main = "x3 = 0")
contour(x1, x2, x3_5, main = "x3 = 5")
contour(x1, x2, x3_neg5, main = "x3 = -5")
contour(x1, x2, x3_10, main = "x3 = 10")
```




```
par(par_default)

#plot 3d
par(mfrow=c(2, 2), mar = c(3, 0, 1, 0))
persp(x1, x2, x3_0,main = "x3 = 0")
persp(x1, x2, x3_5, main = "x3 = 5")
persp(x1, x2, x3_neg5, main = "x3 = -5")
persp(x1, x2, x3_10, main = "x3 = 10")
```




```
par(par_default)

#optim
optim(par = c(0, 0, 0), fn = f)[1:2]
```

```
## $par
## [1] 0.999978292 0.002730698 0.004284640
##
## $value
## [1] 1.876851e-05
```

```
optim(par = c(1, 1, 1), fn = f)[1:2]
```

```
## $par
## [1]  0.9999779414 -0.0001349269 -0.0001927127
##
## $value
## [1] 1.343098e-07

optim(par = c(20, 100, 1e5), fn = f)[1:2]

## $par
## [1]  -4.096157 -11.468532   3.170776
##
## $value
## [1] 16369.8

optim(par = c(20, 100, 1e5), fn = f, method = "BFGS")[1:2]

## $par
## [1] 1.000000e+00 5.252510e-10 8.400998e-10
##
## $value
## [1] 7.099825e-19

optim(par = c(-100, -1000, -1e5), fn = f)[1:2]

## $par
## [1] -4.771779 -6.296438 -6.561802
##
## $value
## [1] 5722.384

optim(par = c(-100, -1000, -1e5), fn = f, method = "BFGS")[1:2]

## $par
## [1] 1.000000e+00 1.880569e-13 3.056140e-13
##
## $value
## [1] 1.215431e-25

#nlm
nlm(f, p = c(0, 0, 0))[1:2]

## $minimum
## [1] 100
##
## $estimate
## [1] 0 0 0

nlm(f, p = c(1, 1, 1))[1:2]

## $minimum
## [1] 1.702065e-08
##
## $estimate
## [1]  9.999995e-01 -8.225859e-05 -1.301257e-04

nlm(f, p = c(20, 100, 1e5))[1:2]
```

```
## $minimum
## [1] 2.140881e-17
##
## $estimate
## [1]  1.000000e+00 -9.779836e-11  2.902313e-10
```

```
nlm(f, p = c(-100, -1000, -1e5))[1:2]
```

```
## $minimum
## [1] 1.674813e-16
##
## $estimate
## [1] 1.000000e+00 1.451395e-09 3.037370e-09
```

I plotted the provided "helical valley" function using constant values of x3 to get slices of x1 and x2. From the plots, it looks like there will be local minima, and there are valleyes throughout the function.

The results from the opimizataions using optim() and nlm() indicate that there are indeed local minima. When using non-ideal non-scaled starting values, both optim and nlm converge to solutions that are incorrect. NLM performs better than optim with "Nelder-Mead" and "BFGS".

3 Consider a censored regression problem. We assume ...

3.1 Design an EM algorithm to estimate the 3 parameters, = (0, 1, 2), taking ...

X : covariates Y : outcome Z : values of censored Y values

Likelihood function:

$$\begin{aligned}
\mathcal{L}(\theta; X, Y, Z) &= \prod_{i=1}^{c} \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}(z_i - (\beta_0 + \beta_1 x_i))^2) \prod_{j=c+1}^{n} \frac{1}{\sqrt{2\sigma^2}} exp(-\frac{1}{2\sigma^2}(y_i - (\beta_0 + \beta_1 x_j))^2) \\
&= (\frac{1}{\sqrt{2\pi\sigma^2}})^n \prod_{i=1}^{c} exp(-\frac{1}{2\sigma^2}(z_i - (\beta_0 + \beta_1 x_i))^2) \prod_{j=c+1}^{n} exp(-\frac{1}{2\sigma^2}(y_i - (\beta_0 + \beta_1 x_j))^2))
\end{aligned}$$

log-Likelihood function:

$$\begin{aligned}
\ell(\theta; X, Y, Z) &= -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{c} (z_i - (\beta_0 + \beta_1 x_i))^2 - \frac{1}{2\sigma^2} \sum_{j=c+1}^{n} (y_i - (\beta_0 + \beta_1 x_j))^2 \\
&= -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{j=c+1}^{n} \sum_{j=c+1}^{n} (y_i - (\beta_0 + \beta_1 x_j))^2 \\
&\quad - \frac{1}{2\sigma^2} \sum_{i=1}^{c} (z_i^2 - 2z_i(\beta_0 + \beta_1 x_i) + (\beta_0 + \beta_1 x_i)^2)
\end{aligned}$$

Expectations and variance:

$$\begin{aligned}
E[\tau^* | X, Y, \theta_t] &= \frac{1}{\sigma_t}(\tau - (\beta_{0,t} + \beta_{1,t} x_i)) \\
E[\rho(\tau^*) | X, Y, \theta_t] &= \frac{\phi(\tau^*)}{(1 - \Phi(\tau^*)^2)} \\
E[z_i | X, Y, \theta_t] &= (\beta_{0,t} + \beta_{1,t} x_i) + \sigma_t \rho(\tau^*) \\
var(z_i | X, Y, \theta_t) &= \sigma_t^2 (1 + \tau^* \rho(\tau^*) - \rho(\tau^*)^2)
\end{aligned}$$

Q function:

$$
\begin{aligned}
Q(\theta|\theta_t) &= E[\ell(\theta; X, Y, Z)|X, Y, \theta_t] \\
&= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=c+1}^{n}(y_i - (\beta_0 + \beta_1 x_j))^2 \\
&\quad + -\frac{1}{2\sigma^2}\sum_{i=1}^{c}(E[z_i^2|X, Y, \theta_t] - 2E[z_i|X, Y, \theta_t](\beta_0 + \beta_1 x_i) + (\beta_0 + \beta_1 x_i)^2) \\
&= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=c+1}^{n}(y_i - (\beta_0 + \beta_1 x_j))^2 \\
&\quad - \frac{1}{2\sigma^2}\sum_{i=1}^{c}(E[z_i|X, Y, \theta_t] - (\beta_0 + \beta_1 x_i))^2 + \sum_{i=1}^{c} var(z_i|X, Y, \theta_t)
\end{aligned}
$$

Partial derivative for β_0

$$
\begin{aligned}
\frac{\partial}{\partial\beta_0}Q(\theta|\theta_t) &= -\frac{1}{2\sigma^2}\sum_{j=c+1}^{n}2(y_i - (\beta_0 + \beta_1 x_j))(-1) - \frac{1}{2\sigma^2}\sum_{i=1}^{c}2(E[z_i|X, Y, \theta_t] - (\beta_0 + \beta_1 x_i))(-1) \\
&= \frac{1}{\sigma^2}(\sum_{j=c+1}^{n}(y_j - \beta_1 x_j) - \sum_{j=c+1}^{n}\beta_0 + \sum_{i=1}^{c}(E[z_i|X, Y, \theta_t] - \beta_1 x_i) - \sum_{i=1}^{c}\beta_o) \\
&= \frac{1}{\sigma^2}(\sum_{j=c+1}^{n}(y_j - \beta_1 x_j) - \sum_{i=1}^{c}(E[z_i|X, Y, \theta_t] - \beta_1 x_i) - n\beta_0) \\
&= 0
\end{aligned}
$$

solve for β_0

$$
\hat{\beta_0} = \frac{1}{n}(\sum_{j=c+1}^{n}(y - \beta_1 x_j) + \sum_{i=1}^{c}(E[z_i|X, Y, \theta_t] - \beta_1 x_i))
$$

Partial derivative for β_1

$$
\begin{aligned}
\frac{\partial}{\partial\beta_1}Q(\theta|\theta_t) &= -\frac{1}{2\sigma^2}\sum_{j=c+1}^{n}2(y_i - (\beta_0 + \beta_1 x_j))(-x_j) - \frac{1}{2\sigma^2}\sum_{i=1}^{c}2(E[z_i|X, Y, \theta_t] - (\beta_0 + \beta_1 x_i))(-x_i) \\
&= \frac{1}{\sigma^2}(\sum_{j=c+1}^{n}x_j(y_j - \beta_0) - \sum_{j=c+1}^{n}\beta_1 x_j^2 + \sum_{i=1}^{c}x_i(E[z_i|X, Y, \theta_t] - \beta_0) - \sum_{i=1}^{c}\beta_1 x_i^2) \\
&= \frac{1}{\sigma^2}(\sum_{j=c+1}^{n}x_j(y_j - \beta_0) + \sum_{i=1}^{c}x_i(E[z_i|X, Y, \theta_t] - \beta_0) - \beta_1\sum_{k=1}^{n}x_k^2) \\
&= 0
\end{aligned}
$$

solve for β_1

$$
\hat{\beta_1} = \frac{1}{\sum_{k=1}^{n}x_k^2}(\sum_{j=c+1}^{n}x_j(y_j - \beta_0) + \sum_{i=1}^{c}x_i(E[z_i|X, Y, \theta_t] - \beta_0))
$$

Partial derivative for σ^2

$$
\begin{aligned}
\frac{\partial}{\partial\sigma^2}Q(\theta|\theta_t) =& -\frac{n}{2}(\frac{1}{2\pi\sigma^2})(2\pi) + \frac{1}{2\sigma^4}\sum_{j=c+1}^{n}(y_j-(\beta_0+\beta_1x_j))^2 \\
&+\frac{1}{2\sigma^4}\sum_{i=1}^{c}(E[z_i|X,Y,\theta_t]-(\beta_0+\beta_1x_i))^2+\frac{1}{2\sigma^4}\sum_{i=1}^{c}var(z_i|X,Y,\theta_t) \\
=&\frac{1}{2\sigma^4}(n\sigma^2+\sum_{j=c+1}^{n}(y_i-(\beta_0+\beta_1x_j))^2) \\
&+\sum_{i=1}^{c}(E[z_i|X,Y,\sigma^t]-(\beta_0+\beta_1x_i))+\sum_{i=1}^{c}var(z_i|X,Y,\theta_t) \\
=&0
\end{aligned}
$$

solve for σ^2

$$
\hat{sigma}^2 = \frac{1}{n}(\sum_{j=c+1}^{n}(y_j-(\beta_0+\beta_1x_j))^2+\sum_{i=1}^{c}(E[z_i|X,Y,\theta_t]-(\beta_0+\beta_1x_i))^2+\sum_{i=1}^{c}var(z_i|X,Y,\theta_t))
$$

The σ^2 estimator is a ratio with a numerator contains the normal sum of squares for the non-censored data and the sum of squares for teh cnesored data, the varaicne of teh imputed censored data using θ^t.

3.2 Propose reasonable starting values for the 3 parameters...

Using observed Y_i values, if

- $\bar{x} = 1/(n-c)\sum_{j=c+1}^{n}$
- $\bar{y} = 1/(n-c)\sum_{j=c+1}^{n} y_j$

then:

$$
\begin{aligned}
\hat{\beta}_{0,0} &= 1/(n-c)\sum_{j=c+1}^{n}(y_j-\hat{\beta}_{1,0}x_j) \\
&\;... \\
&= 1/(n-c)\sum_{j=c+1}^{n}\left(y_j - x_j\frac{\sum_{j=c+1}^{n}x_j(y_i-\bar{y})}{\sum_{j=c+1}^{n}x_j(x_j-\bar{x})}\right)
\end{aligned}
$$

$$
\begin{aligned}
\hat{\beta}_{1,0} =& \frac{1}{\sum_{j=c+1}^{n}x_j^2}\left(\sum_{j=c+1}^{n}(y_i-\hat{\beta}_{0,0})x_j\right) \\
\hat{\beta}_{1,0}\sum_{j=c+1}^{n}x_j^2 =& \sum_{j=c+1}^{n}x_j(y_j-\frac{1}{n-c}\sum_{j=c+1}^{n}(y_j-\hat{\beta}_{1,0}x_j)) \\
=&\left(\sum_{j=c+1}^{c}x_j(y_j-\bar{y})\right)-(\hat{\beta}_{1,0}\sum_{j=c+1}^{n}x_j\bar{x})
\end{aligned}
$$

$$
\hat{\beta}_{1,0}\sum_{j=c+1}^{n}x_j(x_j-\bar{x}) = \sum_{j=c+1}^{n}x_j(y_j-\bar{y})
$$

$$
\hat{\beta}_{1,0} = \frac{\sum_{j=c+1}^{n}x_j(y_j-\bar{y})}{\sum_{j=c+1}^{n}x_j(x_j-\bar{x})}
$$

$$
\hat{\sigma}_0^2 = \frac{1}{n}\sum_{j=c+1}^{n}(y_j-(\hat{\beta}_{0,0}+\hat{\beta}_{1,0}x_j))^2
$$

3.3 Write an R function, with auxiliary functions as needed...

EM function is below.

```
#generate data
source("./ps8.R")

#########
# EM function
##########

em = function(x, y, tau, stop=1000, stopLike=1e-6) { # x: vector of x_i values
    #y = vector of y_i values, with NA for censored data
    #x = observed covariates
    #b0 = beta0
    #b1 = beta1
    #sigma2 = sigma^2
    #ll = log Likelihood
    #tau: threshold for y_i censoring
    #stop: maximum number of iterations through EM algorithm
    #stopLike: diff of loglik of parameters of iterations
        #returns: data frame of (b_0, b_1, s^2, loglik) for each iteration

    #set output df
    results <- data.frame(matrix(NA, nrow=stop, ncol = 4))
    names(results) <-  c("b0", "b1", "sigma2", "ll")

    #init parms
    missing <- is.na(y)
    mod <- lm(y ~ x)
    results[1, ] <- c(mod$coefficients, var(mod$residuals), logLik(mod)[[1]])

    for(i in 2:stop) {

        #E-step: impute censored data
        mu <- results$b0[i - 1] + results$b1[i - 1] * x[missing]
        tau_star <- (tau - mu) / sqrt(results$sigma2[i - 1])
        rho <- dnorm(tau_star) / (1 - pnorm(tau_star))
        y[missing] <- mu + sqrt(results$sigma2[i - 1]) * rho
        var_z <- results$sigma2[i - 1] * (1 + tau_star * rho-rho^2)

        #M-step: re-compute parameters
        mod <- lm(y ~ x)
        results[i, ] <- c(mod$coefficients,
                          var(mod$residuals) + sum(var_z) / length(x),
                          logLik(mod)[[1]])

        #evaluate stopping criteria
        if(abs(results$ll[i] - results$ll[i - 1]) < stop)
        {return(results[1:i,])}
    }

  return(results)
}
```

Let's check consistency of estimated paramters given a range of missing data.

```
########
# evaluate EM
#########

#full data
parms_no_missing <- c(mod$coefficients, var(mod$residuals), logLik(mod)[[1]])
names(parms_no_missing) <- c("b0", "b1", "sigma2", "logLik")

#20% missing y_i
tau_80 <- quantile(yComplete, probs = c(0.80))
y_80 <- yComplete
y_80[y_80 > tau_80] <- NA
em_80 <- em(x, y_80, tau_80)

# 50% missing y_i
tau_50 <- quantile(yComplete)[3]
y_50 <- yComplete
y_50[y_50 > tau_50] <- NA
em_50 <- em(x, y_50, tau_50)

# 80% missing y_i
tau_20 <- quantile(yComplete, probs = c(0.20))
y_20 <- yComplete
y_20[y_20 > tau_20] <- NA
em_20 <- em(x, y_20, tau_20)
```

Let's check results of estimated paramters given a range of missing data.

```
est_parm <- rbind(parms_no_missing,
                      tail(em_80, 1),
                      tail(em_50, 1),
                      tail(em_20, 1))

colnames(est_parm) <- c("beta0", "beta1", "sigma2", "logLikelihood")
rownames(est_parm) <- c("No miss", "20% miss", "50% miss", "80% miss")

require(xtable)
tbl=xtable(est_parm,caption="Parameter estimates for different missingness thresholds with user defined

print(tbl,floating=T,include.rownames=T,caption.placement="top",caption.width="35em")
```

Table 1: Parameter estimates for different missingness thresholds with user defined function.

	beta0	beta1	sigma2	logLikelihood
No miss	0.56	2.77	5.26	-224.40
20% miss	0.46	2.60	3.93	-208.91
50% miss	0.27	1.89	2.02	-173.11
80% miss	-0.24	0.82	0.73	-115.03

It looks like accuracy of estiamtes (i.e. closer to full data estimates) decreaes as amont of missingness increases. This makes sense. The fact that there are better likelihoods for the EM's with more missing data coudl be due to teh fact that were are estimating/imputating values for z_i's, and we may be doing this in a way that reduces variation, and the resulting likelihood will be larger than teh one with non-censored data.

3.4 A different approach to this problem just directly maximizes the ...

```
require(truncnorm)


###########
# Loglike function
##########

loglikeFunc = function(theta, x, y, tau) {

    #theta = c(beta0, beta1, log(sigma2))
    #x = x_i values (vector)
    #y = y_i values, censored==NA (vector)
    #tau = y_i censor threshold

    #estimate loglik for y_i
    #theta <- mod_80_theta
    #x = x
    #y = y_80
    #tau = tau_80

    beta0 = theta[1]
    beta1 = theta[2]
    sigma2 = exp(theta[3])
    missing = is.na(y)
    mu = beta0 + beta1 * x
    loglik_y = sum(dnorm(y[!missing], mean = mu[!missing],
                         sd = sqrt(sigma2),
                         log = T))

    #estimate z_i (missing y_i values)
    tau_star = (tau - mu[missing]) / sqrt(sigma2)
    rho = dnorm(tau_star) / (1 - pnorm(tau_star))
    z = mu[missing] + sqrt(sigma2) * rho
    var_z = sigma2 * (1 + tau_star * rho - rho^2)
    loglik_z = sum(log(dtruncnorm(z, a = tau, mean = mu[missing], sd = sqrt(var_z))))

    return((loglik_y + loglik_z))
}

#optim implementation
    #mut specifiy fnscale = -1 to make optiom maximize our objective function and maximize our likeliho

#20pp missing
mod_80 <- lm(y_80 ~ x)
mod_80_theta <- c(mod_80$coefficients, log(var(mod_80$residuals)))
optim_80 <- optim(mod_80_theta, fn = loglikeFunc, x = x, y = y_80, tau = tau_80,
                  control = list(parscale=c(0.1, 1, 1), fnscale = -1),
                  method="BFGS", hessian = T)
```

```r
#80 pp missing
mod_20 <- lm(y_20 ~ x)
mod_20_theta  <- c(mod_20$coefficients, log(var(mod_20$residuals)))
optim_20 <- optim(mod_20_theta, fn = loglikeFunc, x = x, y = y_20, tau = tau_20,
                  control = list(parscale=c(0.1, 1, 1), fnscale = -1),
                  method="BFGS", hessian = T)


res_optim <- rbind("No miss" = parms_no_missing,
                   "20% miss" = c(optim_80$par, optim_80$value),
                   "80% miss" = c(optim_20$par, optim_20$value))
```

```r
#print latex tables of rsults of optim function
require(xtable)
tbl=xtable(res_optim,caption="Parameter estimates for different missingness thresholds with optim().",c

print(tbl,floating=T,include.rownames=T,caption.placement="top",caption.width="35em")
```

Table 2: Parameter estimates for different missingness thresholds with optim().

	b0	b1	sigma2	logLik
No miss	0.56	2.77	5.26	-224.40
20% miss	0.69	2.44	1.35	-204.51
80% miss	-0.34	0.60	-0.86	-73.95

Overall, optim() performed better than my function, and had larger likelihoods than my function, but the results were fairly similar. At missingness levels of 20% and 80% and both my function and optim() were similar, with more missingness reducing acuracy of estimates. There was more functuation betwen σ^2 estimates between optim and my fucntion. I didn't find any difference in performance using parscale arguments.