Homework 2

Hannah Zmuda

10/27/2020

Question 1

Bisection Method

```r
bisection <- function(f, a, b, nMax, tol)
{
  #initiate the a and b value, assume intervals will be proper
  iteration <- 0
  rootArray <- nMax

  #check bounds
  if(f(a) == 0.0){
    return(a)
  }
  if(f(b) == 0.0){
    return(b)
  }
  # Begin method's loop
  for (i in 1:nMax){
    c <- (a + b)/2 #Calc the midpoint
    rootArray[i] <- c
    if(f(c) != 0) {
      #TRUE: f(c) > tol AND i <=NMAX
      if((abs(f(c)) > tol)) {
        if(sign(f(c)) == sign(f(a))) {
          a <- c
          b <- b
        }
        else {
          a <- a
          b <- c
        }
        c <- (a + b)/2
        root <- tail(rootArray,n = 1)
        result <- list('root' = root, 'iterations' = rootArray)
      }
      else {
        #the f(c) is within the range of tolerance
        break
      }
    }
    else {
      #FALSE: f(c) is a root
      break
    }
  }
  return(list('root' = root, 'iterations' = rootArray))
}

fcn <- function(x){sqrt(x)-cos(x)}
bmMe <- bisection(fcn, 0, 2, 10, 1e-5)
bmMe$iterations
```

```
##  [1] 1.0000000 0.5000000 0.7500000 0.6250000 0.6875000 0.6562500 0.6406250
##  [8] 0.6484375 0.6445312 0.6425781
```

```r
bmMe$iterations[3]
```

```
## [1] 0.75
```

Newton-Raphson Method

```r
newton <- function(f, dx, a, b, inital, nMax, tol){
  #set initial value
  x0 <- inital
  rootArray <- nMax

  #Check bounds
  if(f(a) == 0.0){
    return(a)
  }
  if(f(b) == 0.0){
    return(b)
  }

  #begin loop for loop method
  for (i in 1:nMax) {
    x1 = x0 - (f(x0)/dx(x0))
    rootArray[i] <- x1
    if (abs(x1 - x0) <= tol){
      root <- tail(rootArray,n = 1)
      result <- list('root' = root, 'iterations' = rootArray)
      return(result)
    }
    x0 <- x1
  }
}

f <- function(x){sqrt(x)-cos(x)}
```

```r
dx <- function(x){0.5*(x^(-0.5)) + sin(x)}
newMe <- newton(f, dx, 0, 2, 1, 10, 1e-5)
newMe$root
```

```
## [1] 0.6417144
```

```r
newMe$iterations
```

```
## [1] 0.6573182 0.6417461 0.6417144 0.6417144
```

```r
newMe$iterations[3]
```

```
## [1] 0.6417144
```

The Newton-Raphuson Method finds the root within the three iterations, compared to the Bisection Method. The Bisection Method found 0.75 when the tolerance is 1e-5. For the same tolerance, the Newton's Method found the root within 3 to 4 iterations (0.6417144), depending on the number of iterations. The Newton's Method is more effective because an initial value is given before entering the loop and the updated value utilizes the slop of the function (dx) instead of simply going by midpoint x-values.

Question 2

Part a: Deriving the Newton-Raphson Method

In the problem, we are told we can use the Poisson process assumption so we can have the likelihood function:

$$L(N|\lambda_i) = \sum_{i=1}^{n} \frac{\lambda^{N_i} e^{-\lambda}}{N!} \tag{1}$$

and because $\lambda_i = \alpha_1 b_{i1} + \alpha_2 b_{i2}$ we can substitute λ_i into Eq (1):

$$L(N|\alpha_1, \alpha_2) = \sum_{i=1}^{n} \frac{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^{N_i} e^{-(\alpha_1 b_{i1} + \alpha_2 b_{i2})}}{N!} \tag{2}$$

In order to find the parameters α_1 and α_2 we can use the Newton-Raphson update which needs to become Eq (3):

$$\begin{bmatrix} \alpha_1(t+1) \\ \alpha_2(t+1) \end{bmatrix} = \begin{bmatrix} \alpha_1(t) \\ \alpha_2(t) \end{bmatrix} - \frac{L\prime}{L\prime\prime} \tag{3}$$

In order to get Eq (3), we first need to get the log likelihood of Eq (2):

$$l(N|\alpha_1, \alpha_2) = \sum_{i=1}^{n} N_i \ln(\alpha_1 b_{i1} + \alpha_2 b_{i2}) - \sum_{i=1}^{n} \alpha_1 b_{i1} + \alpha_2 b_{i2} - \sum_{i=1}^{n} \ln(N!) \tag{4}$$

We can then use Eq (4) and take the partial first derivative in regard to both parameters α_1 and α_2:

$$l\prime(N|\alpha_1, \alpha_2) = \begin{bmatrix} \sum_{i=1}^{n} \frac{N_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i1} \\ \sum_{i=1}^{n} \frac{N_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i2} \end{bmatrix} \tag{5}$$

To get the double derivative we use the Hessian matrix

$$l\prime\prime(N|\alpha_1, \alpha_2) = \begin{bmatrix} \frac{\partial^2 l}{\partial \alpha_1^2} & \frac{\partial^2 l}{\partial \alpha_1 \partial \alpha_2} \\ \frac{\partial^2 l}{\partial \alpha_2^2} & \frac{\partial^2 l}{\partial \alpha_2 \partial \alpha_1} \end{bmatrix} \tag{6}$$

$$l\prime\prime(N|\alpha_1, \alpha_2) = \begin{bmatrix} \sum_{i=1}^{n} -\frac{N_i b_{i1}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & \sum_{i=1}^{n} -\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \\ \sum_{i=1}^{n} -\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & \sum_{i=1}^{n} -\frac{N_i b_{i2}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \end{bmatrix} \tag{7}$$

with this we can get the equation to be used in the Newton-Raphson update to get a final equation

$$\begin{bmatrix} \alpha_1(t+1) \\ \alpha_2(t+1) \end{bmatrix} = \begin{bmatrix} \alpha_1(t) \\ \alpha_2(t) \end{bmatrix} - \begin{bmatrix} \sum_{i=1}^{n} -\frac{N_i b_{i1}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & \sum_{i=1}^{n} -\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \\ \sum_{i=1}^{n} -\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} & \sum_{i=1}^{n} -\frac{N_i b_{i2}^2}{(\alpha_1 b_{i1} + \alpha_2 b_{i2})^2} \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{n} \frac{N_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i1} \\ \sum_{i=1}^{n} \frac{N_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i2} \end{bmatrix} \tag{8}$$

Part b: Deriving the Fisher Scoring Method

Similar to part (a), we will use the log likelihood to find the equation to find the parameters α_1 and α_2. Instead of only taking the second derivative, we will take the variance of the first derivative to get the Fisher Information. This output will be a two by two matrix as well but instead is the Covariance matrix instead of the Hessian matrix:

$$\begin{bmatrix} Var(\sum_{i=1}^{n} \frac{N_i b_{i1}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i1}) & Covariance \\ Covariance & Var(\sum_{i=1}^{n} \frac{N_i b_{i2}}{\alpha_1 b_{i1} + \alpha_2 b_{i2}} - \sum_{i=1}^{n} b_{i2}) \end{bmatrix} \tag{9}$$

Or we could take the (negative) expectation of the second derivative of the log likelihood function.

$$I(\alpha_1,\alpha_2) = -E[l\prime\prime(N|(\alpha_1,\alpha_2))] = \begin{bmatrix} \sum_{i=1}^{n} E[\frac{N_i b_{i1}^2}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})^2}] & \sum_{i=1}^{n} E[\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})^2}] \\ \sum_{i=1}^{n} E[\frac{N_i b_{i1} b_{i2}}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})^2}] & \sum_{i=1}^{n} E[\frac{N_i b_{i2}^2}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})^2}] \end{bmatrix} \tag{10}$$

$$I(\alpha_1,\alpha_2) = -E[l\prime\prime(N|(\alpha_1,\alpha_2))] = \begin{bmatrix} (\sum_{i=1}^{n} \frac{b_{i1}^2}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})}) & (\sum_{i=1}^{n} \frac{b_{i1} b_{i2}}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})}) \\ (\sum_{i=1}^{n} \frac{b_{i1} b_{i2}}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})}) & (\sum_{i=1}^{n} \frac{b_{i2}^2}{(\alpha_1 b_{i1}+\alpha_2 b_{i2})}) \end{bmatrix} \tag{11}$$

Equation 11 then simplifies down to the Fisher Scoring Approach:

$$\begin{bmatrix} \alpha_1(t+1) \\ \alpha_2(t+1) \end{bmatrix} = \begin{bmatrix} \alpha_1(t) \\ \alpha_2(t) \end{bmatrix} + I(\theta^t)^{-1} l\prime(\theta^t) \tag{12}$$

Part c and d: Implementing Newton and Fisher Methods

```
#import data set from Givens et al.
data.oil <- read.table("oilspills.dat",header = TRUE)

#likelihood function
l <- function(N, theta.old){
  result <- sum(N*log(theta.old)) - sum(theta.old) - sum(log(factorial(N)))
  return(result)
}
#derivative of the likelihood function
dl <- function(N, b1, b2, theta.old){
  result1 <- sum((N*b1)/(theta.old[1]*b1 + theta.old[2]*b2)-b1)
  result2 <- sum((N*b2)/(theta.old[1]*b1 + theta.old[2]*b2)-b2)
  output <- as.numeric(list(result1,result2))
  return(matrix(data = output, ncol = 1))
}
#double derivative of the likelihood function
d2l <- function(N, b1, b2, theta.old){
  result11 <- -1*sum((N*b1^2)/(theta.old[1]*b1 + theta.old[2]*b2)^2)#1st row, 1st col
  result12 <- -1*sum((N*b1*b2)/(theta.old[1]*b1 + theta.old[2]*b2)^2)#non-principal components
  result22 <- -1*sum((N*b2^2)/(theta.old[1]*b1 + theta.old[2]*b2)^2)#2nd row, 2nd col
  output <- as.numeric(list(result11,result12,result12,result22))
  return(matrix(data = output, nrow = 2, ncol = 2, byrow = TRUE))
}
#Fisher Information
I <- function(N, b1, b2, theta.old){
  result11 <- sum((b1^2)/(theta.old[1]*b1 + theta.old[2]*b2))#1st row, 1st col
  result12 <- sum((b1*b2)/(theta.old[1]*b1 + theta.old[2]*b2))#non-principal components
  result22 <- sum((b2^2)/(theta.old[1]*b1 + theta.old[2]*b2))#2nd row, 2nd col
  output <- as.numeric(list(result11,result12,result12,result22))
  return(matrix(data = output, nrow = 2, byrow = TRUE))
}

#Newton's Method
new.oil <- function(N,b1,b2){
  n <- 200
  i <- 1
  theta.old <- matrix(1,2,1)
```

```r
  tol <- 10^-10
  for(i in 1:n){
    theta.new = theta.old - (solve(d2l(N,b1,b2,theta.old)) %*% dl(N,b1,b2,theta.old))
    root <- theta.new
    current.tol <- sum(abs(theta.new - theta.old))
    if(current.tol <= tol){
      return(list("iteration" = i,"root" = root))
    }
    theta.old <- theta.new
  }
}
#fisher function
fish.oil <- function(N,b1,b2){
  n <- 200
  i <- 1
  theta.old <- matrix(1,2,1)
  tol <- 10^-10
  for(i in 1:n){
    theta.new = theta.old + (solve(I(N,b1,b2,theta.old)) %*% dl(N,b1,b2,theta.old))
    root <- theta.new
    current.tol <- sum(abs(theta.new - theta.old))
    if(current.tol <= tol){
      return(list("iteration" = i,"root" = root))
    }
    theta.old <- theta.new
  }
}

#main body of code (i.e. no function definitions)
N <- data.oil$spills
b1 <- data.oil$importexport
b2 <- data.oil$domestic
output.new <- new.oil(N,b1,b2)
output.fish <- fish.oil(N,b1,b2)
#How can I best compare performance of these two functions?
#Use convergence map like in the Givens et al. example?
print(output.new$iteration)
```

```
## [1] 5
```

```r
print(output.fish$iteration)
```

```
## [1] 20
```

```r
#part d: Calculating the standard error
I.error <- function(N, b1, b2, theta.old){
  result11 <- sum((b1^2)/(theta.old[1]*b1 + theta.old[2]*b2))#1st row, 1st col
  result12 <- sqrt(sum((b1*b2)/(theta.old[1]*b1 + theta.old[2]*b2)))#non-principal components
  result22 <- sum((b2^2)/(theta.old[1]*b1 + theta.old[2]*b2))#2nd row, 2nd col
  output <- as.numeric(list(result11,result12,result12,result22))
  return(matrix(data = output, nrow = 2, byrow = TRUE))
}
```

```r
newSE = sqrt(solve(I.error(N,b1,b2,output.new$root)))
```

```
## Warning in sqrt(solve(I.error(N, b1, b2, output.new$root))): NaNs produced
```

```r
print(newSE)
```

```
##           [,1]     [,2]
## [1,] 0.2563221      NaN
## [2,]       NaN 0.369917
```

```r
fishSE = sqrt(solve(I.error(N,b1,b2,output.fish$root)))
```

```
## Warning in sqrt(solve(I.error(N, b1, b2, output.fish$root))): NaNs produced
```

```r
print(fishSE)
```

```
##           [,1]     [,2]
## [1,] 0.2563221      NaN
## [2,]       NaN 0.369917
```

Part e: Quasi-newton Method

```r
data.oil <- read.table("oilspills.dat", header = TRUE)
#Use log likelihood equation
l <- function(dataset, p){
  result <- (-1)*sum(dataset$spills*log(p[1]*dataset$importexport + p[2]*dataset$domestic) - (p[1]*datas
  return(result)
}
#Gradient output
output <- function(dataset, p){
  result1 <- sum((dataset$spill*dataset$importexport)/(p[1]*dataset$importexport + p[2]*dataset$domesti
  result2 <- sum((dataset$spill*dataset$domestic)/(p[1]*dataset$importexport + p[2]*dataset$domestic)-da
  output <- as.numeric(list(result1,result2))
  return((-1)*matrix(data = output, ncol = 1))
}
#Standard error function
I.error <- function(N, b1, b2, theta.old){
  result11 <- sum((b1^2)/(theta.old[1]*b1 + theta.old[2]*b2))#1st row, 1st col
  result12 <- sqrt(sum((b1*b2)/(theta.old[1]*b1 + theta.old[2]*b2)))#non-principal components
  result22 <- sum((b2^2)/(theta.old[1]*b1 + theta.old[2]*b2))#2nd row, 2nd col
  output <- as.numeric(list(result11,result12,result12,result22))
  return(matrix(data = output, nrow = 2, byrow = TRUE))
}
#optim function
optim.output <- optim(par = c(1,1),fn = l,gr = output,dataset = data.oil,method = "BFGS",hessian = T)
print(optim.output)
```

```
## $par
## [1] 1.0971525 0.9375546
##
## $value
## [1] 23.25506
##
## $counts
## function gradient
##       17        5
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##          [,1]     [,2]
## [1,] 16.66034 9.099290
## [2,]  9.09929 8.219997
```

```
#standard error from optim function
optim.error <- sqrt(solve(optim.output$hessian))
```

```
## Warning in sqrt(solve(optim.output$hessian)): NaNs produced
```

```
print(optim.error)
```

```
##           [,1]      [,2]
## [1,] 0.3896128       NaN
## [2,]       NaN 0.5546758
```

Question 3

##Part a based on the problem, we know we are given an observed data set $X = (X_1, X_2, ..., X_n)$ where $X_i \sim \alpha N(\mu_1, \sigma_1^2) + (1-\alpha)N(\mu_2, \sigma_2^2)$. with this, and knowing we have a two-component mixture model, we can derive the EM algorithm to find $\hat{\theta}$ where $\theta = (\alpha, \mu_1, \mu_2, \sigma_1, \sigma_2)$. For the Q function we have:

$$Q(\theta|\hat{\theta}) = E[log(L(\theta|X))]$$

$$Q(\theta|\hat{\theta}) = \Sigma_{i=1}^{n} log(\alpha N(X_i, \mu_1, \sigma_1) + (1-\alpha)N(X_i, \mu_2, \sigma_2))$$

From here, we can say that the Maximum Likelihood extimate for the parameters μ an dσ is as follows:

$$\hat{\mu_j} = \frac{\Sigma_i X_i P(\theta_j|X_i)}{\Sigma_i P(\theta_j|X_i)}$$

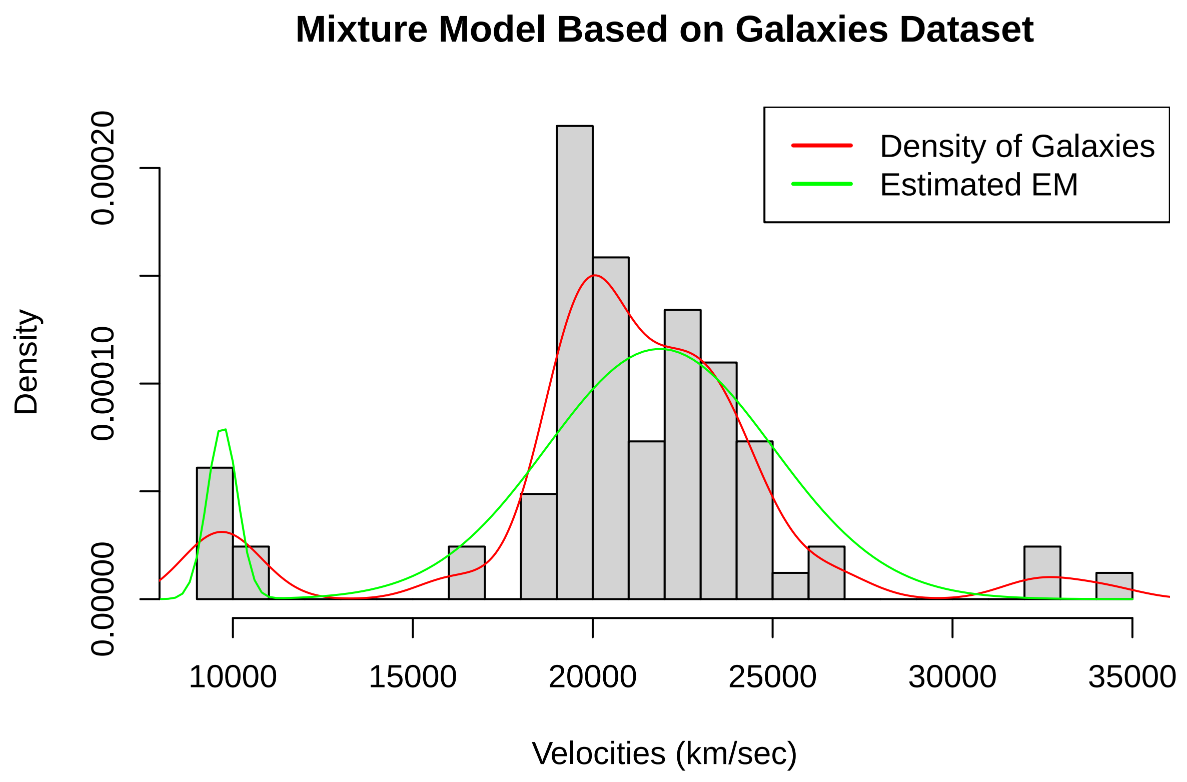
$$\hat{\sigma_j} = \frac{\Sigma_i (X_i - \mu_j)^2 P(\theta_j|X_i)}{\Sigma_i P(\theta_j|X_i)}$$

##Part b

```r
#call the galaxies dataset
data(galaxies)
x <- galaxies
n <- length(x)
#set.seed(200)
#make an estimation for the mu, sigma, and alpha variables. Can use k clustering based on the remark
kCluster <- kmeans(x,2)$cluster
mu1 <- mean(x[kCluster == 1])
mu2 <- mean(x[kCluster == 2])
sigma1 <- sd(x[kCluster == 1])
sigma2 <- sd(x[kCluster == 2])
alpha <- sum(kCluster == 1)/length(kCluster)#Make alpha based on the first mixture
#Other variables
i <- 2 #number of mixture models (two-mixture model)
tol <- 1e-10 #tolerance for the EM Algorithm
#Calculate Q
#initialize Q
Q <- 0
Q[2] <- sum(log(alpha*dnorm(x,mu1,sqrt(sigma1)))) + sum(log((1-alpha)*dnorm(x,mu2,sqrt(sigma2))))
#E step: Compute Q(theta | theta^t) where theta = (alpha,mu1,mu2,sig1,sig2)
while(abs(Q[i]-Q[i-1])>=tol){
  #Find the conditional probability for each mixture model
  mix1 <- alpha*dnorm(x,mu1,sigma1)
  mix2 <- (1-alpha)*dnorm(x,mu2,sigma2)
  totalMix <- mix1 + mix2
  cp1 <- mix1/totalMix#Conditional Probability for mixture 1
  cp2 <- mix2/totalMix#Conditional probability for mixture 2
  #M step
  alpha <- sum(cp1)/n
  mu1 <- sum(x*cp1)/sum(cp1)
  mu2 <- sum(x*cp2)/sum(cp2)
  sigma1 <- sqrt(sum(((x-mu1)^2)*cp1)/sum(cp1))
  sigma2 <- sqrt(sum(((x-mu2)^2)*cp2)/sum(cp2))
  #update probability values
  cp1 <- alpha
  cp2 <- 1-alpha
  #update counter
  i <- i + 1
  #Return E-step, unless stopping criteria has been met
  Q[i] <- sum(log(totalMix))
}
theta <- list("alpha" = alpha, "mu1" = mu1, "mu2" = mu2, "sigma1" = sigma1,"sigma" = sigma2)
print(theta)
```

```
## $alpha
## [1] 0.08518786
##
## $mu1
## [1] 9709.316
##
## $mu2
## [1] 21863.57
##
## $sigma1
## [1] 422.1317
##
## $sigma
## [1] 3144.631
```

```
hist(x, prob = T, breaks = 20, xlab = "Velocities (km/sec)", main = "Mixture Model Based on Galaxies Da
lines(density(x), col = "red")
xfit <- seq(8000,35000,200)
EMEstimate <- (alpha * dnorm(xfit,mu1,sigma1)) + ((1-alpha) * dnorm(xfit,mu2,sigma2))
lines(xfit, EMEstimate, col = "green",ylim = max(EMEstimate))
legend('topright', col = c("red","green"), lwd = 2, legend = c("Density of Galaxies", "Estimated EM"))
```



Because there are multiple local maxima in the equation, we cannot use this method for MLE.

Question 4

Using a,b parameters

Because some of the data in the problem is omitted (censored), we can say we only have the observed data set, not the complete data set. To find the Q function for the EM algorithm (and MCEM algorithm).

Step 1: Find the Joint pdf of the observed data

We are given a Weibull distribution with the density function $f(y) = aby^{b-1}exp(-ay^b)$ for $0 < y$ and for the parameters a and b. We also have the observed data x_i where $x_i = (min(y_i, c_i), \delta_i)$

$$\begin{cases} \delta_i = 1, \quad X_i = Y_i = f(x_i, \delta_i = 1) = abx_i^{b-1}exp(-ax_i^b)\int_{x_i}^{\infty} g(c)dc \\ \delta_i = 0, \quad X_i = C_i = f(x_i, \delta_i = 0) = exp(-ax_i^b)g(x_i) \end{cases}$$

We can then combine the two cases into the following:

$$[abx_i^{b-1}exp(-ax_i^b)\int_{x_i}^{\infty} g(c|\eta)dc]^{\delta_i}[exp(-ax_i^b)g(x_i|\eta)]^{1-\delta_i}$$

Because we do not care about the function g or η, we can focus on the other parts of the equation:

$$\Pi_{i=1}^{n} a^{\delta_i} b^{\delta_i} x_i^{\delta_i(b-1)} exp(-ax_i^b)$$

$$(ab)^{\Sigma\delta_i}\Sigma_{i=1}^{n}(x_i^{\delta_i(b-1)}exp(-ax_i^b))$$

Step 2: Calculate the observed log likelihood

With the joint pdf simplified and including the censor indicator, δ, we can now calculate the log likelihood. This can be found by $l(\theta|x, \delta) = \Sigma_{i=1}^{n} log(f(x_i, \delta_i|\theta))$. Expanding this equation out we get:

$$l(a, b|x_i, \delta_i) = \Sigma_{i=1}^{n}\delta_i(log(a) + log(b)) - \Sigma_{i=1}^{n} ax_i^b + log(\Sigma_{i=1}^{n} x_i^{\delta_i(b-1)})$$

Step 3: Derive the Q-function

For our last step to derive the Q function, we will take the expectation of the log likelihood function:

$$Q(\theta|\theta(t)) = E[l(\theta|x_i)|y, \theta^{(t)}]$$

$$= \Sigma_{i=1}^{n} E[n(log(a) + log(b)) - ax_i^b|y_i, \theta^{(t)} + (b-1)log(x_i)|y, \theta^{(t)}]$$

$$= n(log(a) + log(b)) - a\Sigma_{i=1}^{n} E[x_i^b|y_i, \theta^{(t)}] + (b-1)\Sigma_{i=1}^{n} E[(log(x_i|y_i, \theta^{(t)})]$$

As we can see from this equation, we have x_i in multiple parts of the Q function. Moreover we have two different outputs based on the observed data. If we have $x_i = y_i$ then $\delta_i = 1$ while if $x_i = c_i$ then $\delta = 0$ and $L(\theta, x_i|x_i > c_i, \theta^{(t)})$. Because of this, the Q function is difficult to compute analytically. One work around for this is to calculate an approximation of the Q function, $\hat{Q}$.

$$\hat{Q}(\theta|\theta^{(t)}) = \frac{1}{m^{(t)}}\Sigma_{j=1}^{m^{(t)}} L(\theta|x_j)$$

As we can see, the previous equation is much easier to numerically compute. However, as $m^{(t)}$ becomes increasingly smaller the estimator approximation becomes less accurate. We can also use Conditional Maximization steps in the M step. This would simplify the estimation of a and b in the Q function more easily than the Q function using the EM algorithm.