STP 540 Final Project

Demetri Papakostas, Veeresh Yalasangi & Wynetta Herrera

In this project, we compare the Expectation-Maximization (EM) algorithm and the Monte Carlo Expectation-Maximization (MCEM) algorithm for censored exponential data.

INTRODUCTION

Suppose we begin with a joint model

$$f(z, x|\theta)$$

with both observed x and latent z variables. The presence of latent variables can certainly complicate finding the MLE of parameter θ. In such cases, we utilize the EM algorithm to obtain the MLE of θ. Each iteration of the EM algorithm consists of two steps: the expectation E-step and the maximization M-step.

- In the E step, the missing data are estimated given the observed data and current estimate of the parameters using the conditional expectation

- In the M step, the likelihood function is maximized with the assumption that the missing data are known using the estimate of the missing data from the E-step in place of the actual missing data

Iterates of the EM algorithm are set up as such:

E-step

$$Q(\theta|\theta^t) = E(log(f(z, x|\theta))$$

where E is over $Z|x, \theta^t$.

M-step

$$(\theta^{t+1}) = \underset{\theta}{\text{argmax}} Q(\theta|\theta^t)$$

This is repeated until convergence.

Some cases, however, do not admit a closed form solution for the Q function and so the expectation in the E-step cannot be done analytically. In these cases, we turn to MCEM wherein the expectation is replaced with a Monte Carlo approximation for the E-step. We get iid draws from $z|x, \theta^t$ and then compute a Monte Carlo approximation of the expectation:

$$Q(\theta|\theta^t) = E(log(f(z,x|\theta)) \approx \frac{1}{m}\sum(f(z,x|\theta)$$

where $z_j \sim Z|x,\theta^t, iid$

We then move to the M-step to maximize $\hat{Q}^{(t+1)}(\theta|\theta^t)$ and continue to iterate until convergence.

THE DATASET

The dataset we use is given by the "Computational Statistics" textbook by G.H. Givens and J.A. Hoeting. We have thirty iid observations where $Yi \sim Exp(\lambda)$, some of which are right censored. The observed data are thus $x = (x_1, \ldots x_n)$ where $x_i = (min(y_i, c_i), \delta_i)$, the c_i are the censoring levels, and $\delta_i = 1$ if $y_i \leq c_i$ and $\delta_i = 0$ otherwise. We will attempt to estimate parameter λ from this data using both the EM and MCEM algorithms. We will initate both from $\lambda^0 = 0.5042$, which is the mean of all 30 values disregarding censoring.

EM FOR DATASET

To find parameter λ for our exponential dataset using the ordinary EM algorithm, we compute the conditional expectation via

$$\ell(\lambda|Y) = n\ log\ \lambda - \lambda\sum Y_i$$

$$= n\ log\ \lambda - \lambda\sum_{i=1}^{n}[\,y_i\delta_i + c_i(1+\delta_i)] - \frac{C\delta}{\lambda^{(t+1)}}$$

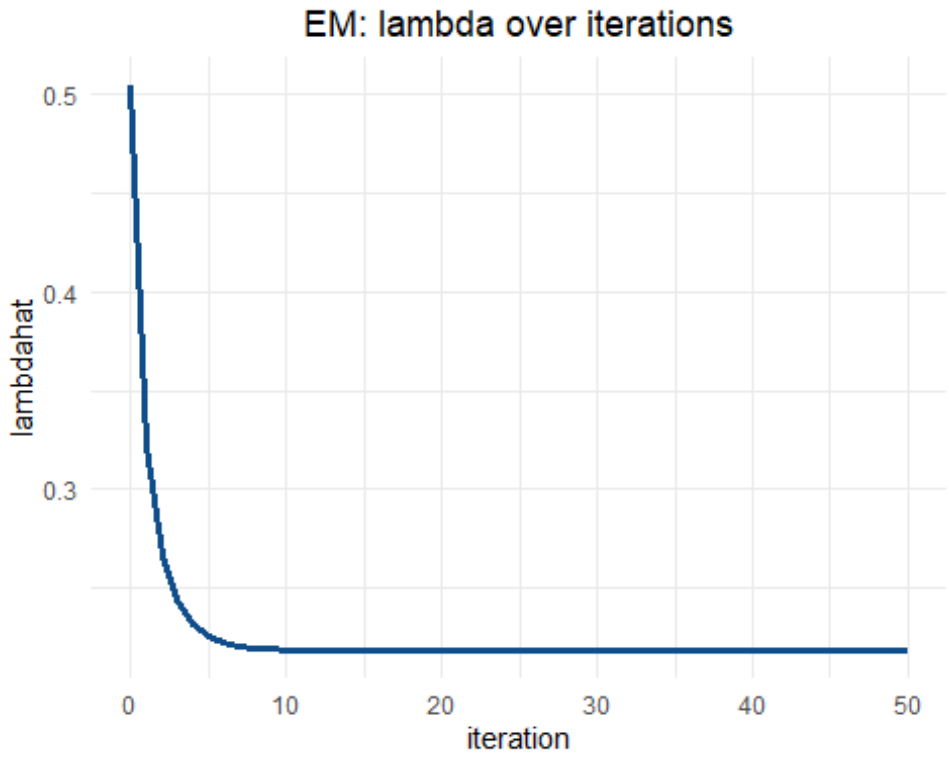
where $C = \sum_{i=1}^{n}(1-\delta_i)$ denotes the number of censored cases. This expression is then maximized to get the following update to parameter λ:

$$\lambda^{(t+1)} = \frac{n}{\sum_{i=1}^{n} x_i + \frac{C}{\lambda^t}}$$

.

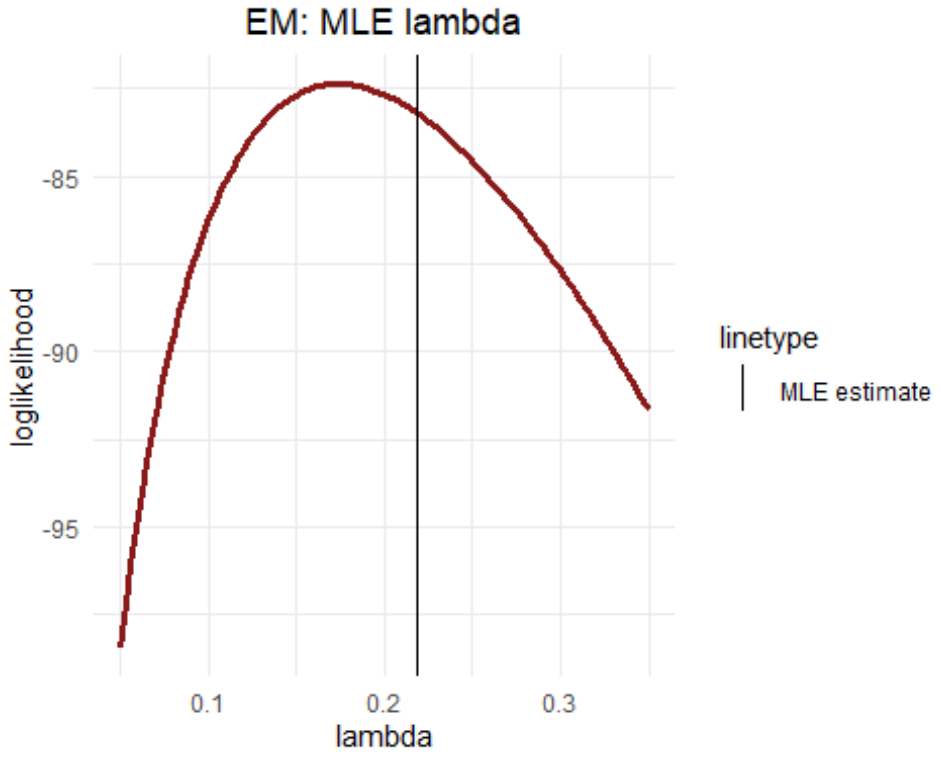
```
library(dplyr, warn.conflicts=F, quietly=T)
library(ggplot2)
data=read.table("censoreddata.dat",
           header=TRUE, skip=0)
n=nrow(data)
lambda0=0.5042

lambda0=data%>%filter(deltai==1)%>%summarize(mean(obsvi))#$obsvi)
lambda0
lambdalist=c()
for (i in 1:n){
  lambdalist[[i+1]]=n/(sum(data$obsvi)+data$ci/lambdalist[[i]])
}
```

```r
data
data%>%filter()%>%summarize(mean(ci), mean(obsvi), mean(Yi))

niter=50
#initial guess
lambdadat=c(0.5042, rep(0,niter))


lambda=1  #for now
for (i in 1:niter){

  expvalue=n*log(lambda)-lambda*sum(data$Yi+data$ci*(1-data$deltai))-(sum(1-
data$deltai)*lambda)/(lambda0)
  lambdadat[[i+1]]=n/(sum(data$obsvi)+(sum(1-data$deltai))/lambdadat[[i]])

}
data.frame(iteration=seq(0, niter), lambdahat=lambdadat)%>%
   ggplot(aes(x=iteration, y=lambdahat))+ggtitle("EM: lambda over
iterations")+geom_line(lwd=1.2,
color='dodgerblue4')+theme_minimal()+theme(plot.title = element_text(hjust =
0.5))
```




```r
####Expectation Step####
emanalytic=function(lambda){
for (i in 1:niter){

  expvalue=n*log(lambda)-lambda*sum(data$Yi+data$ci*(1-data$deltai))-(sum(1-
```

```
data$deltai)*lambda)/(lambda0)
  lambdadat[[i+1]]=n/(sum(data$obsvi)+(sum(1-data$deltai))/lambdadat[[i]])

}
  return(c(unlist(expvalue),min(lambdadat)  ))
}
N=100
loglike=c(rep(0,N))

lambavarlist=seq(0.05,.35, length.out=100)

for (i in 1:N){
loglike[[i]]=emanalytic(lambavarlist[[i]])[1]
lambdadat2=emanalytic(lambavarlist[[i]])[2]
}
data.frame(lambda=lambavarlist, loglikelihood=loglike)%>%
  ggplot(aes(x=lambda, y=loglikelihood))+ggtitle("EM: MLE
lambda")+geom_line(lwd=1.2,
color='firebrick4')+geom_vline(aes(xintercept=lambdadat2, linetype='MLE
estimate'))+theme_minimal()+theme(plot.title = element_text(hjust = 0.5))
```



With the EM algorithm, we find that MLE $\hat{\lambda}$ is indeed near the given textbook value of 0.2185 and that convergence occurs fairly quickly.

MCEM FOR DATASET

With MCEM, the expression representing the E step for our exponential dataset becomes:

$$\hat{Q}^{(t+1)}(\lambda|\lambda^{(t)}) = n\ log\ \lambda\ -\ \frac{\lambda}{m^{(t)}}\sum_{j=1}^{m^t} Y_j^T\ 1$$

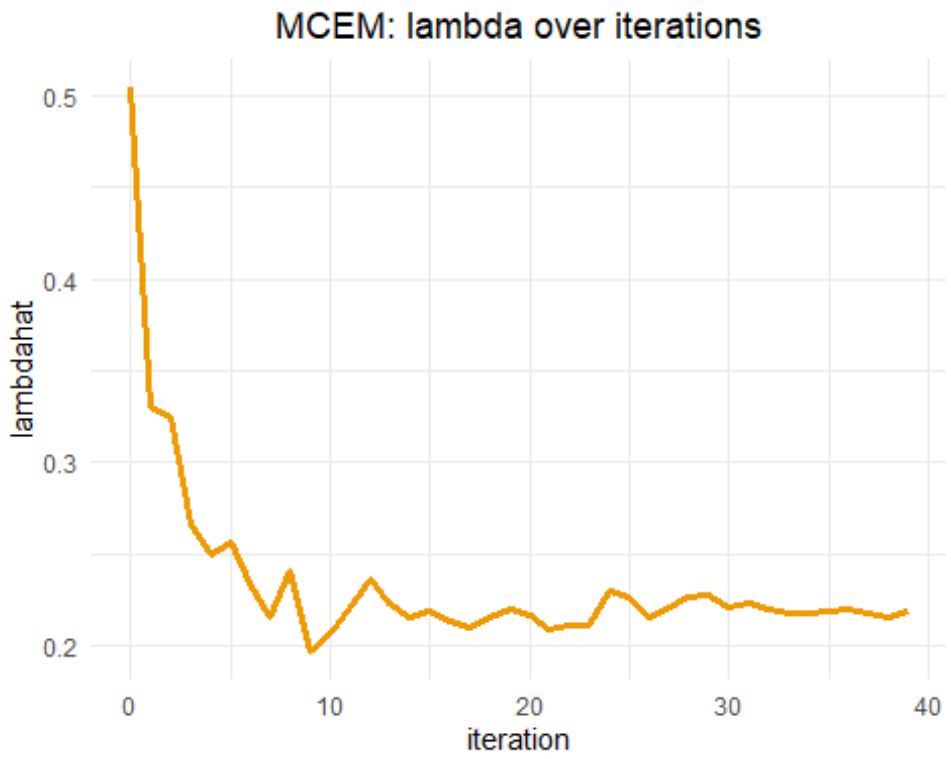
where 1 is a vector of ones and Y_j is the j_{th} completed dataset comprising the uncensored data and simulated data $Z_j = (Z_{j1}, \ldots, Z_{jC})$ from $m^{(t)}$ draws with $Z_{jk} - c_k \sim iid\ Exp(\lambda^{(t)})$ for $k = 1, \ldots, C$ to replace the censored values.

We can then update MCEM with the following equation:

$$\lambda^{(t+1)} = \frac{n}{\sum_{j=1}^{m^t} Y_j^T\ 1/m^t}$$

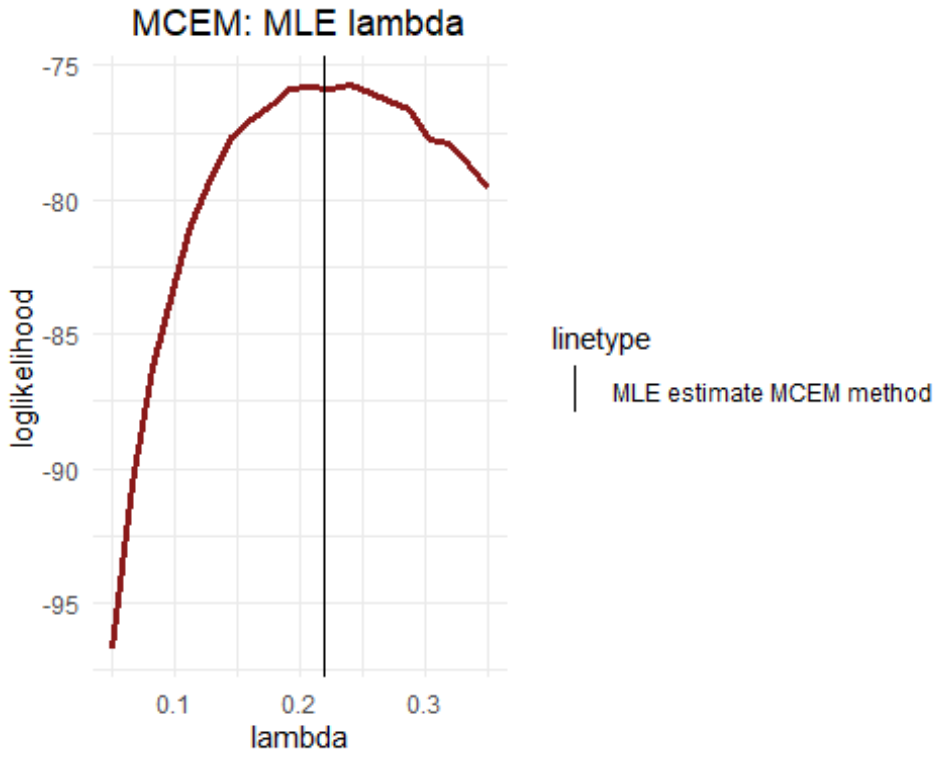
```
mfun=function(t){
  return(5^(1+floor(t/10)))
}

sumfun<- function(x, start, end){
  return(sum(x[start:end]))
}

simdata=function(lambdadat,seed){
  set.seed=seed
  ifelse(data$deltai==0,rexp(10, lambdadat)+data$ci, data$obsvi)
}

onevec=rep(1, n)
niter=39
#initial guess
lambdadat=c(0.5042, rep(0.01,niter))




emmontecarlo=function(niter){
Y=c()
for (i in 1:niter){

   for (j in 1:mfun(i)){

  #we make our combined data frame with the simulated data and old together.
  #sample.int ensures we get random draws.
  Y[[j]]= simdata(lambdadat[[i]], sample.int(10000))
   }
  #merge the datasets, so we have m monte carlo draws of the set
```

```
  #i.e. when 1<=i<10, we have 5 Yj's, each of which we sum
  Yj=do.call(cbind, Y)
  #print(Yj)
  #lambdadat[[i+1]]=n/((mfun(i)*t(simdata)%*%onevec)/(mfun(i)))

  #here, we sum of the different Yj up until the monte carlo iteration we are
on...
  # since we store them as a new matrix, we sum the columns up to mfun(i)
times


  lambdadat[[i+1]]=(n*mfun(i))/(sum(Yj[,1:mfun(i)]))



  #print(simdatalist2)
}
  return( lambdadat)
}


niter=39
#initial guess
lambdadat=c(0.5042, rep(0,niter))


lambda=1  #for now
for (i in 1:niter){

  expvalue=n*log(lambda)-lambda*sum(data$Yi+data$ci*(1-data$deltai))-(sum(1-
data$deltai)*lambda)/(lambda0)
  lambdadat[[i+1]]=n/(sum(data$obsvi)+(sum(1-data$deltai))/lambdadat[[i]])

}
data.frame(iteration=seq(0, niter), lambdahat=emmontecarlo(niter),
lambdanalytic=lambdadat)%>%
  ggplot()+ggtitle("MCEM: lambda over iterations")+geom_line(aes(x=iteration,
y=lambdahat),lty=1, lwd=1.2,
color='orange2')+theme_minimal()+theme(plot.title = element_text(hjust =
0.5))
```




```
####Expectation Step####
emmclikelihood=function(niter, lambda)
  {
Y=c()
for (i in 1:niter){

   for (j in 1:mfun(i)){

  #we make our combined data frame with the simulated data and old together.
  #sample.int ensures we get random draws.
  Y[[j]]= simdata(lambdadat[[i]], sample.int(10000))
   }
  #merge the datasets, so we have m monte carlo draws of the set
  #i.e. when 1<=i<10, we have 5 Yj's, each of which we sum
  Yj=do.call(cbind, Y)

  #here, we sum of the different Yj up until the monte carlo iteration we are
on...
  # since we store them as a new matrix, we sum the columns up to mfun(i)
times

  expvalue=n*log(lambda)-(lambda/mfun(i))*(sum(Yj[,1:mfun(i)]))

}
  return( expvalue)
}
```

```
N=20
loglike=c(rep(0,N))

lambavarlist=seq(0.05,.35, length.out=20)

for (i in 1:N){
loglike[[i]]=emmclikelihood(niter, lambavarlist[[i]])
}
data.frame(lambda=lambavarlist, loglikelihood=loglike)%>%
  ggplot(aes(x=lambda, y=loglikelihood))+ggtitle("MCEM: MLE
lambda")+geom_line(lwd=1.2,
color='firebrick4')+geom_vline(aes(xintercept=lambdadat2, linetype='MLE
estimate MCEM method'))+theme_minimal()+theme(plot.title = element_text(hjust
= 0.5))
```



Using $m^{(t)} = 5^{1+[\frac{t}{10}]}$ as in the textbook, we find that for MCEM, MLE $\hat{\lambda}$ is indeed near the given value of 0.2185. Convergence does eventually occur but we do find that the estimate bounces around the true maximum as iterations proceed, which is to be expected with MCEM.

EM VERSUS MCEM

Lastly, we compare $\hat{\lambda}$ over the iterations for both the EM and MCEM algorithms in a graph similar to Figure 4.2 in the textbook. We find both converge to the same MLE $\hat{\lambda}$ value, ~ 0.2185. The analytical EM solution converges quickly whereas the one for MCEM (with its

MC approximation in the E-step) bounces around before convergence. The EM algorithm is certainly a powerful tool that allows us to find MLE of parameters in statistical models that depend on unobserved latent variables. The MCEM algorithm gives us a way to be able to use this powerful tool even in cases when the E-step cannot be calculated analytically.

```
data.frame(iteration=seq(0, niter), lambdahat=emmontecarlo(niter),
lambdanalytic=lambdadat)%>%
  ggplot()+ggtitle("EM vs MCEM: lambda over
iterations")+geom_line(aes(x=iteration, y=lambdahat),lty=2, lwd=1,
color='orange2')+
  geom_line(aes(x=iteration, y=lambdanalytic), lty=1, lwd=1,
color='dodgerblue4')+theme_minimal()+theme(plot.title = element_text(hjust =
0.5))
```