University of Edinburgh
School of Mathematics
Incomplete Data Analysis

Supporting Materials for Lecture 7

I start with the code to reproduce the toy example in slide 9. We create a function that implements the EM algorithm. This function takes as input a starting value for θ, say θ_0, and the value ϵ used for the stopping criterion: $|\theta^{(t)} - \theta^{(t-1)}| < \epsilon$. The variable `diff` in the code below is just to control whether the convergence criterion is met or not. Although we have started setting `diff`=1, any value greater than ϵ would work. The variable `theta.old` stores the value from the previous iteration, so that we can compute $|\theta^{(t)} - \theta^{(t-1)}|$.

```
rm(list=ls())

toyex=function(theta0,eps){
diff=1
theta=theta0
while(diff>eps){
theta.old=theta
theta=2*theta/(5*theta+1)
diff=abs(theta-theta.old)
}
return(theta)
}

toyex(10,0.00001)

## [1] 0.200006
```

I now provide the code for the genetic linkage example (slide 12). It is pretty much similar to the one given above, only withexample specific modifications.

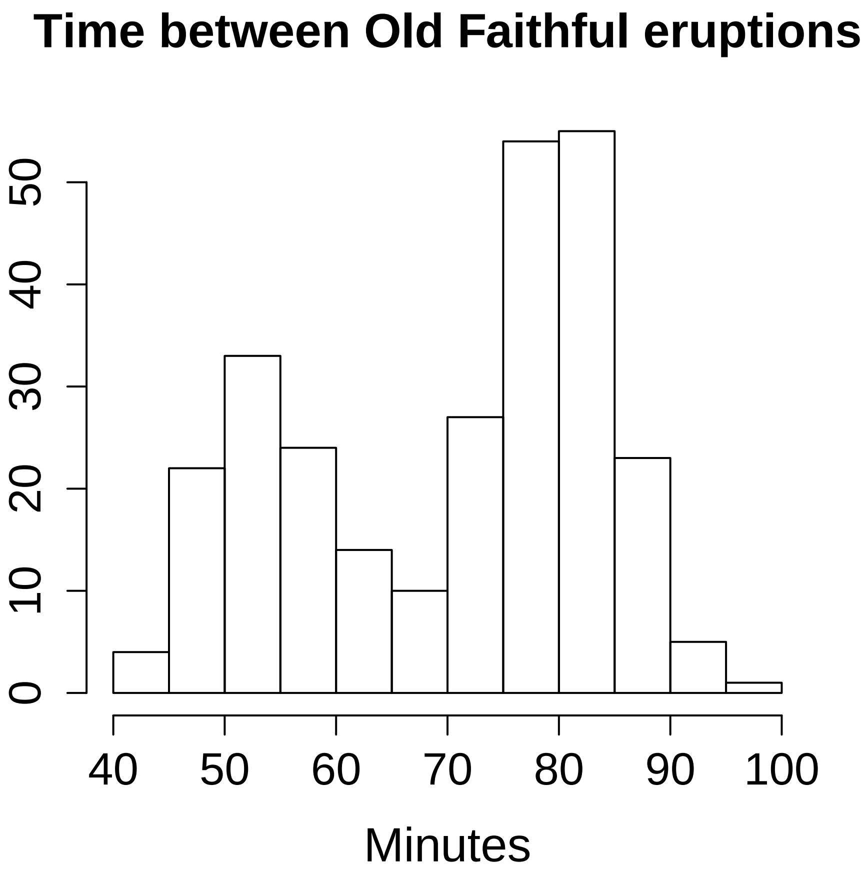
```
multi=function(y,theta0,eps){
n=sum(y); diff=1
theta=theta0
while(diff>eps){
theta.old=theta
#E step
zt=y[1]*0.5/(0.5+0.25*theta)
#M step
theta=(y[1]+y[4]-zt)/(n-zt)
diff=abs(theta-theta.old)
}
return(theta)
}

y=c(125,18,20,34)
multi(y=y,0.5,0.00001)

## [1] 0.6268207
```

Below, I provide the code to implement the EM algorithm for a mixture of two normal distributions.

```r
em.mixture.two.normal=function(y,theta0,eps){
n=length(y)
theta=theta0
p=theta[1]; mu1=theta[2]; sigma1=theta[3]; mu2=theta[4]; sigma2=theta[5]
diff=1
while(diff>eps){
theta.old=theta
#E-step
ptilde1=p*dnorm(y,mean=mu1,sd=sigma1)
ptilde2=(1-p)*dnorm(y,mean=mu2,sd=sigma2)
ptilde=ptilde1/(ptilde1+ptilde2)
#M-step
p=mean(ptilde)
mu1=sum(y*ptilde)/sum(ptilde)
sigma1=sqrt(sum(((y-mu1)^2)*ptilde)/sum(ptilde))
mu2=sum(y*(1-ptilde))/sum(1-ptilde)
sigma2=sqrt(sum(((y-mu2)^2)*(1-ptilde))/sum(1-ptilde))
theta=c(p,mu1,sigma1,mu2,sigma2)
diff=sum(abs(theta-theta.old))
}
return(theta)
}
```

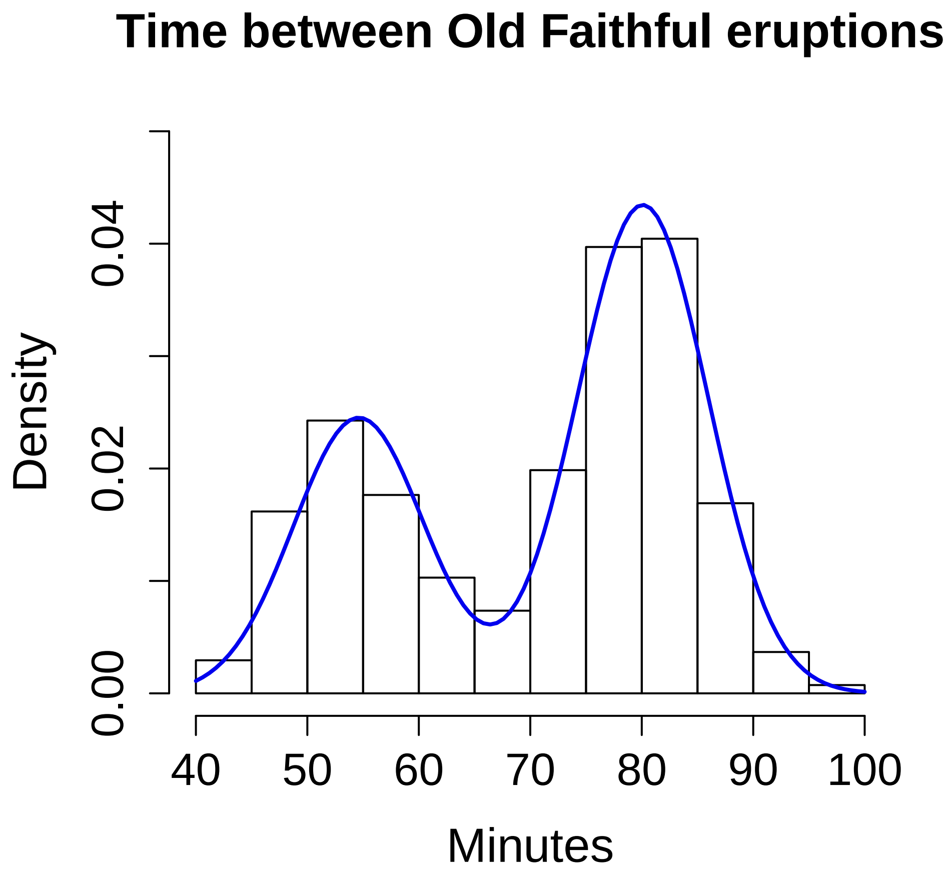
We will now load old faithful dataset.

```r
data(faithful)
attach(faithful)
hist(waiting, main="Time between Old Faithful eruptions",xlab="Minutes",
     ylab="", cex.main=1.5, cex.lab=1.5, cex.axis=1.4)
```



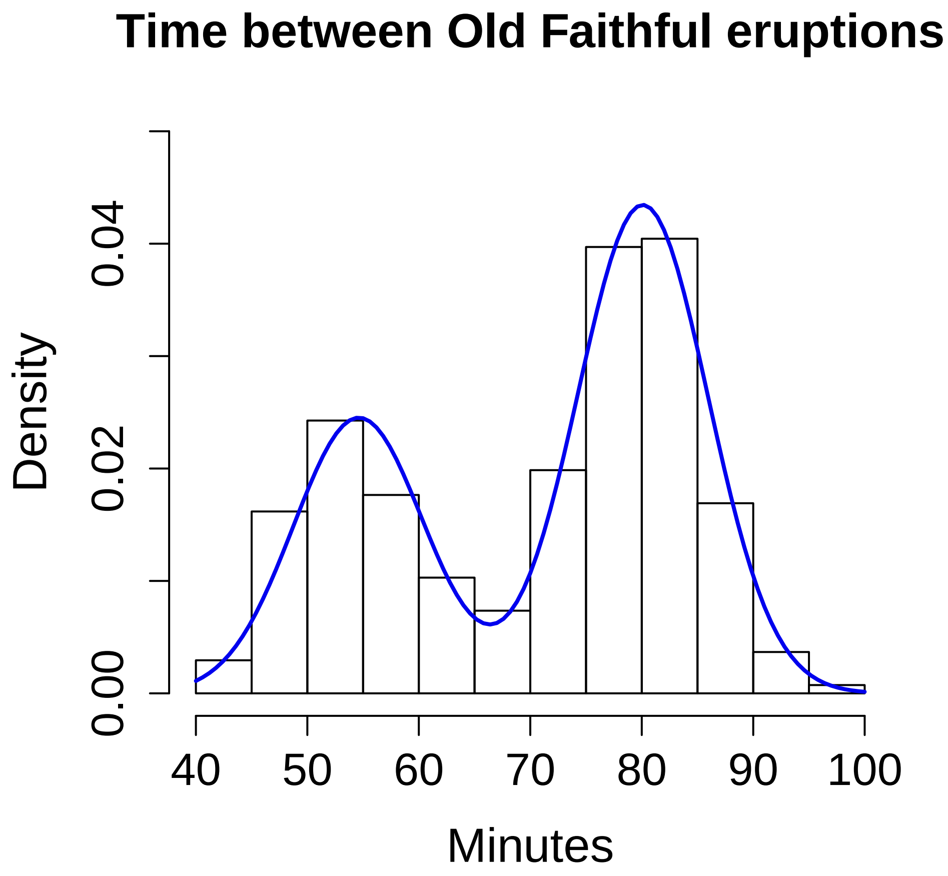
Lastly, we apply the developed function to the data.

```
res=em.mixture.two.normal(y=waiting,c(0.4,40,4,90,4),0.00001)
p=res[1]; mu1=res[2]; sigma1=res[3]; mu2=res[4]; sigma2=res[5]

hist(waiting, main="Time between Old Faithful eruptions",xlab="Minutes",
     ylab="Density", cex.main=1.5, cex.lab=1.5, cex.axis=1.4,freq=F,ylim=c(0,0.05))
curve(p*dnorm(x,mu1,sigma1)+(1-p)*dnorm(x,mu2,sigma2),add=TRUE,lwd=2,col="blue2")
```



We can alternatively use the package `mixtools`. The code follows below.

```
require(mixtools)
res1=normalmixEM(waiting,lambda=0.4,mu=c(40,90), sigma=c(4,4))

## number of iterations= 7

p=res1$lambda[1]
mu1=res1$mu[1]; mu2=res1$mu[2]
sigma1=res1$sigma[1]; sigma2=res1$sigma[2]

hist(waiting, main="Time between Old Faithful eruptions",xlab="Minutes",
     ylab="Density", cex.main=1.5, cex.lab=1.5, cex.axis=1.4,freq=F,ylim=c(0,0.05))
curve(p*dnorm(x,mu1,sigma1)+(1-p)*dnorm(x,mu2,sigma2),add=TRUE,lwd=2,col="blue2")
```


Time between Old Faithful eruptions
Density
0.00
0.02
0.04
40
50
60
70
80
90
100
Minutes